НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

# ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 Інженерія програмного забезпечення

на тему: «Система автоматизованого генерування VR-сцен»

Виконав: студент 4 курсу, групи ТВ-61

_____Солодкий Дмитро Олександрович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____проф. Отрох С.І._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____к.т.н., доц. Шалденко О.В._____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

**Національний технічний університет України**
**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

# ЗАВДАННЯ
**на дипломну роботу студенту**

Солодкий Дмитро Олександрович
(прізвище, ім’я, по батькові)

1. Тема роботи «Система автоматизованого генерування VR-сцен» .

Керівник роботи проф. Отрох Сергій Іванович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25”травня 2020р. № 1168-с

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи платформа Unity, мова програмування C#

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Провести аналіз системи автоматизованого генерування VR-сцен, існуючих систем. Обрати методи для реалізації та візуалізації запропонованого функціоналу і розробити архітектуру системи. Вибрати та реалізувати засоби розробки програмного забезпечення і створити інтуїтивно зрозумілий інтерфейс.

5. Перелік ілюстративного матеріалу

Титульний слайд, актуальність, постановка задачі, аналіз програмних засобів, діаграма прецедентів, діаграма прецедентів-2, діаграма класів, аналіз реалізації, інструменти розробки, демонстрація відеоматеріалу, висновки

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | к.т.н., доц. Шалденко О.В. | | |

7. Дата видачі завдання "_11_"__жовтня__201_9_ р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітки |
|---|---|---|---|
| 1. | Затвердження теми роботи | 17.10.19 | |
| 2. | Вивчення та аналіз задачі | 13.04.20 – 26.04.20 | |
| 3. | Розробка архітектури та загальної структури системи | 27.04.20 | |
| 4. | Розробка структур окремих підсистем | 28.04.20 | |
| 5. | Програмна реалізація системи | 29.04.20 – 12.05.20 | |
| 6. | Оформлення пояснювальної записки | 11.05.20 – 31.05.20 | |
| 7. | Захист програмного продукту | 26.05.20 | |
| 8. | Передзахист | 10.06.20 | |
| 9. | Захист | 16.06.20 | |

Студент _____ Солодкий Д.О
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Отрох С.І
(підпис) (прізвище та ініціали,)

# АНОТАЦІЯ

**Структура та обсяг бакалаврської дипломної роботи**

Дипломна робота складається зі вступу, п’яти розділів, висновку, переліку посилань з 17 найменувань, трьох додатків, і містить 42 рисунка.

Загальний обсяг дипломної роботи бакалавра складає 72 сторінки, з яких перелік посилань займає 2 сторінки та додатки – 13 сторінок.

**Метою дослідження** є аналіз існуючих програмних застосунків в області автоматизованого генерування сцен, розробці засобів конструювання або завантаження 3D моделей приміщень та демонстрації дослідження цих моделей у VR сцені штучним інтелектом.

**Методи та результати дослідження.** Розв’язання поставлених задач вирішувалось поступово. За основу реалізації було обрано платформу Unity, яка ідеально підходить до вирішення поставленої задачі. Тому сцена конструювання або завантаження готової 3D моделі приміщення була створена за допомогою базових інструментів обраної платформи.

Функціонал завантаження визначених в іншій програмі камер було створено за допомогою десеріалізації формату JSON. Цей вибір був обумовлений зручністю та популярністю даного формату обміну даними та обраною платформою.

Крім того, було створено авторський алгоритм генерування та запікання VR-сцени в межах реального часу за допомогою інструментів, які мають змогу працювати в режимі роботи додатку- пакету Google VR SDK та компоненту NavMeshSurface.

**Ключові слова**: *АВТОМАТИЗОВАНЕ ГЕНЕРУВАННЯ СЦЕН, VR-СЦЕНИ, ВІРТУАЛЬНА РЕАЛЬНІСТЬ, КОНСТРУЮВАННЯ ПРИМІЩЕННЯ, ЗАПІКАННЯ NAVMESH В МЕЖАХ РЕАЛЬНОГО ЧАСУ, ЗАВАНТАЖЕННЯ МОДЕЛІ ПРИМІЩЕННЯ, АВТОМАТИЗАЦІЯ ПРИКЛАДНИХ ПРОЦЕСІВ.*

## ABSTRACT

**The structure and content of the bachelor's thesis**

Thesis consists of an introduction, five chapters, a conclusion, a list of 17 references, three appendices, and contains 42 figures.

The total volume of the bachelor's thesis is 72 pages, including the list of links - 2 pages and appendices - 13 pages.

**The purpose of the study** is to analyze existing software applications in the field of automated scene generation, development of tools for designing or loading 3D models of premises and demonstration of the study of these models in VR by artificial intelligence.

**Research methods and results.** The solutions to the tasks were realized gradually. The Unity platform, which is ideal for solving the problem, was chosen as the basis for implementation. Therefore, the scene of designing or loading the finished 3D model of the room was created using the basic tools of the selected platform.

The download functionality of the cameras defined in another program was created by deserializing the JSON format. This choice was due to the convenience and popularity of this data exchange format and the chosen platform.

In addition, an author's algorithm for generating and baking a VR scene in real time was created with the help of tools that can work in the mode of Google VR SDK and NavMeshSurface component application operation.



**Keywords**: *AUTOMATED GENERATION OF SCENES, VR-SCENES, VIRTUAL REALITY, CONSTRUCTION OF PREMISES, RUNTIME NAVMESH BAKING, UPLOADING OF THE MODEL PREMISES, AUTOMATION OF APPLICATIONS.*

# ЗМІСТ

# ВСТУП

Давно існує така галузь, як комп'ютерна графіка. За своєю сутністю, вона є маніпуляцією візуальним контентом. З моменту становлення, вона використовувалась для візуалізації дво-/тривимірного простору та обробки зображень. За цей час, векторна графіка, значною мірою сприяла розвитку взаємодії людини з комп'ютером з точки зору демонстрації теоретичних результатів і практичних застосувань. Оскільки перевірка теоретичних та практичних результатів продовжують ставати більш доступними, зростає необхідність створення додатків, які автоматизують візуалізацію результатів обробки даних. Саме ця галузь і дала початок формуванню нині відомої технології як віртуальна реальність(Virtual Reality).

Віртуальна реальність – це один із можливих способів візуалізації та демонстрації методом залучення особи до гри чи творчого процесу. В межах віртуальної реальності можна спостерігати за віртуальним середовищем як за реальним світом. Одягаючи гарнітуру віртуальної реальності, ви можете бути залученим до гри або потрапити у приміщення, де можна маніпулювати об’єктами та предметами віртуального світу.

Технологія віртуальної реальності сьогодні користується широким попитом у багатьох сферах. VR використовується для інтерактивного залучення покупців товарів і послуг, де користувач вступає у взаємодію з контентом. Можливості віртуального світу дозволяють переносити людину в різні виміри, простори і географічні точки нашого світу, побачити те, що приховано від очей в реальності.

Віртуальна реальність являє собою складний інструмент, який може бути використаний в межах платформи(середовища) для розробки. Середовище розробки, доступне лише програмісту, адже, лише він може керувати його функціями. Користувач, в свою чергу, може отримувати лише візуальний контент, який процедурно був розроблений програмістом.

Інтеграція віртуальної реальності активно застосовується сьогодні в різних бізнес-сегментах: споживчий, комерційні ринки, ринок розваг, подорожей, система

освіти активно впроваджує технології VR в процес навчання[8]. За допомогою віртуальної реальності можна презентувати різні продукти ще до початку їх створення. Це може бути віртуальна презентація зовнішнього вигляду, планувань, дизайну. VR-технології допомагають побачити майбутні об'єкти в умовах, максимально наближених до реальності. Це допомагає уникнути можливих фінансових втрат при виправленні помилок, допущених в проекті, в ході його створення. Віртуальна реальність допомагає побачити недоробки в проектах, опрацювати ергономічні питання, оцінити проміжні етапи вирішення проблем. Фізичні макети можна легко замінити на віртуальні, у проектувальників і замовників з'являється можливість вивчити об'єкти, які проектуються в VR-окулярах, щоб переконатися, що проект відповідає всім вимогам. Існує безліч можливостей інтеграції, проте, більшість процесів, в основі яких закладений загальний механізм, вимагають автоматизації у побудові сцен з такою технологією. Зростає і актуальність проблеми автоматизації побудови цих механізмів, за допомогою яких можна проводити розрахунки необхідні не тільки розробнику, але і користувачу.

Дана проблема є важливим об’єктом для дослідження. Тому, було вирішено реалізувати програмний продукт для автоматизації прикладного процесу, де користувач може створити або завантажити модель реального приміщення(наприклад продуктового магазину) та отримати результат перевірки аналізу приміщення на основі встановлених в ньому камер.

В даній роботі було використано базові методи маніпуляції з ігровими об’єктами та їх компонентами і було досліджено та застосовано інструменти віртуальної реальності.

# 1. ПОСТАНОВКА ЗАДАЧІ МОДЕЛЮВАННЯ ТА СТВОРЕННЯ ПРОЦЕСІВ АВТОМАТИЗОВАНОГО ГЕНЕРУВАННЯ СЦЕН І ПЕРЕВІРКИ ПРИМІЩЕНЬ

На сьогодні, великою проблемою стала систематизація інформації та виділення серед неї корисних для застосування знань. Існує безліч сфер діяльності де лише через складність автоматизації процесів використовується праця людей. Тому, це дало поштовх для застосування комп'ютерів замість людей в тих професіях, в яких можна автоматизувати процес з меншими похибками і більшою ефективністю роботи [6].

Таким чином, досліджувана система може бути використана в різних сферах, як звичайним користувачем, так і програмістом для перевірки взаємодій 3D-об'єкта і приміщення.

Серед таких сфер діяльності можна виділити роботу у галузі безпеки. Вирішення більшості задач у даній сфері залежить від взаємодії з відеопотоком, що вимагає правильного встановлення камер спеціалістом. Проблеми, які вирішують спеціалісти такого плану – встановлення та налаштування камер, створення їх індивідуального плану установки, де враховується тип приміщення та можливі сліпі зони(зони приміщення, які не потрапляють під діапазон камер). Кожна задача вимагає точності розрахунків, з мінімальною затратою часу та мінімальною кількістю помилок.

Тому, для автоматизації та спрощення основних прикладних процесів у галузі безпеки, було запропоновано дослідити методи моделювання та створення процесів автоматизованого генерування сцен і перевірки приміщення на сліпі зони та можливість їх застосування для спрощення роботи фахівця галузі безпеки. Запропонована система дасть змогу швидко та точно побудувати чи завантажити приміщення у масштабі 1:1, завантажити чи встановити камери та візуалізувати всі процеси у віртуальній реальності.

В результаті дослідження, було вирішено створити програмний продукт,

основним функціоналом якого є:

— забезпечення користувача сценою-конструктором, в якій можна будувати чи завантажити власний 3D-план приміщення, обстежувати приміщення і доповнювати новими елементами.

— забезпечення користувача визначенням положення камери відносно об’єктів у створеному/завантаженому приміщенні.

— надати можливість користувачеві завантажувати JSON файл, який зберігає в собі дані про визначені в іншій програмі камери. На основі цих даних встановити камери.

— автоматизоване генерування сцени на основі сцени-конструктора.

— надати можливість користувачеві використовувати функціонал VR-гарнітури.

— автоматизоване створення агентів(штучного інтелекту), який буде досліджувати приміщення.

— забезпечення користувача стеженням за агентом, для визначення доцільності встановлення камер.

— забезпечення користувача зручним графічним інтерфейсом, який може працювати як на комп’ютері, так і на мобільному телефоні.

## Висновок до розділу 1

Задача моделювання та створення процесів автоматизованого генерування сцен і перевірки приміщення на сліпі зони полягає в проведенні аналізу взаємодії користувача із системою для забезпечення відповідної реакції запропонованого програмного комплексу. Даний програмний продукт надає змогу автоматизувати роботу створення VR-сцени, перевірити приміщення шляхом запуску агентів(штучного інтелекту) та візуалізувати ці процеси зручним чином.

# 2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ АВТОМАТИЗОВАНОГО ГЕНЕРУВАННЯ І ВІЗУАЛІЗАЦІЇ СЦЕН У VR

На даний момент, аналогічні програмні продукти мають лише спільну логіку у конструюванні 3D моделей будівель і візуалізації їх у VR не генеруючи нових сцен. Ця логіка розрахована лише на візуалізацію створеного інтер'єру(що мало чим схоже на запропоновану систему). Найпопулярніші програми побудови плану і візуалізації інтер'єру: " RoomSketcher", "Sweet Home" та громіздкі програми роботи з графікою.

Інструменти Unity(Assets) по завантаженні і оброці цих моделей були виявлені лише платними, окремих програм взагалі не було виявлено. Аналогічні програми по виявленні сліпих зон існують лише для автомобілів і мають іншу логіку(Gazer,Maserati,Ford).

Найбільш схожий результат можна отримати за допомогою програмного забезпечення сім'ї "Autocad". Проте, такий програмний продукт громіздкий, вимагає навички професіонала, важко автоматизувати процес, який не зв'язаний з конструюванням.

Таким чином, лише схожий результат можна отримати, і те, тільки у конструюванні 3д плану будівлі або ж використовуючи програмне забезпечення окремо.

## 2.1 Аналіз існуючих програмних засобів

У процесі аналізу існуючих рішень, які були зазначені вище, було виявлено, що на даний момент такі системи є досить складними, результат не можна отримати в повному обсязі і в одній програмі.

Кожна з приведених програмних засобів має недоліки, які суттєво ускладнюють роботу користувача з системою.

— “RoomSketcher”(рисунок 2.1) – це платна веб-система з відкритим вихідним кодом для моделювання інтер'єру , архітектурної візуалізації житлових просторів і плану будинку у 2d, 3d, VR. Система може будувати та імпортувати тривимірні моделі в форматах obj, dae, 3ds, lws. Крім того, система дає можливість експорту плану в форматах svg і pdf, експорту тривимірного виду в форматі obj. Недоліком даної системи для поставленої задачі можна вважати, що вона платна, розрахована лише на платформу комп’ютерів, і використовується лише для побудови інтер’єра.

Рисунок 2.1 — Інтерфейс RoomSketcher

— Програмні продукти сім’ї “Autodesk” – це дво- і тривимірні системи автоматизованого проектування. Для роботи з цим програмним забезпеченням необхідні навички і досвід. Можна отримати 3D модель будівлі, але за наявністю навичок. Можна лише побудувати необхідну модель, проте використовувати її доведеться в інших програмах, спеціалізованих на необхідних прикладних задачах.

3ds Max(Рисунок 2.2) – професійне програмне забезпечення для 3D-моделювання, анімації і візуалізації при створенні ігор і проектуванні. 3ds Max у своєму розпорядженні володіє великими засобами для створення різноманітних за формою і складності тривимірних комп'ютерних моделей, реальних чи фантастичних об'єктів навколишнього світу, з використанням різноманітних технік і

механізмів. Ця система використовується лише професіоналами і лише на платній основі, як і інші програмні продукти сім'ї Autodesk.

AutoCAD(Рисунок 2.3) – дво- і тривимірна система автоматизованого проектування і креслення Ця система знайшла широке застосування в машинобудуванні, будівництві, архітектурі та інших галузях промисловості. AutoCAD включає в себе повний набір інструментів для комплексного тривимірного моделювання. AutoCAD дозволяє отримати високоякісну візуалізацію моделей за допомогою системи рендеринга mental ray. Також в програмі реалізовано управління тривимірною печаткою і підтримка хмар точок (дозволяє працювати з результатами 3D-сканування ).

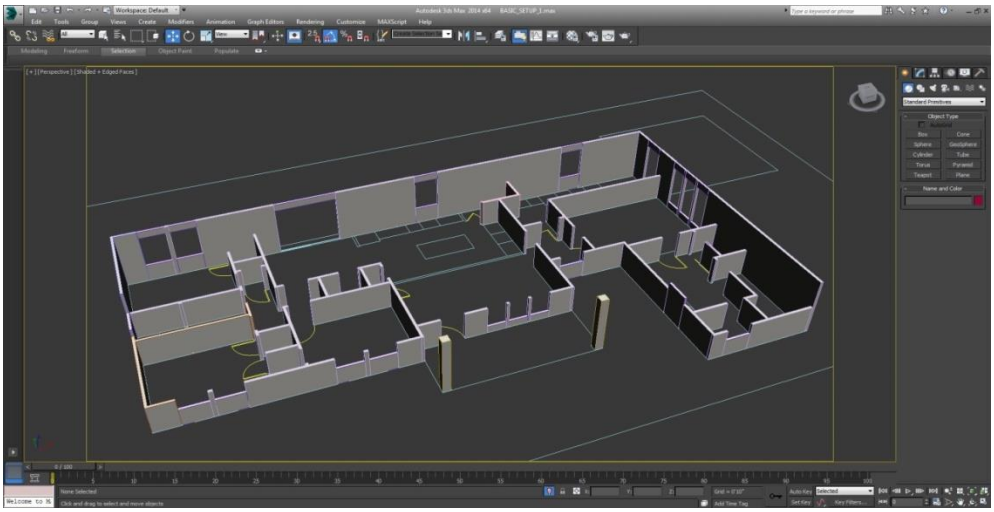

Рисунок 2.2 — Інтерфейс Autodesk 3ds Max.

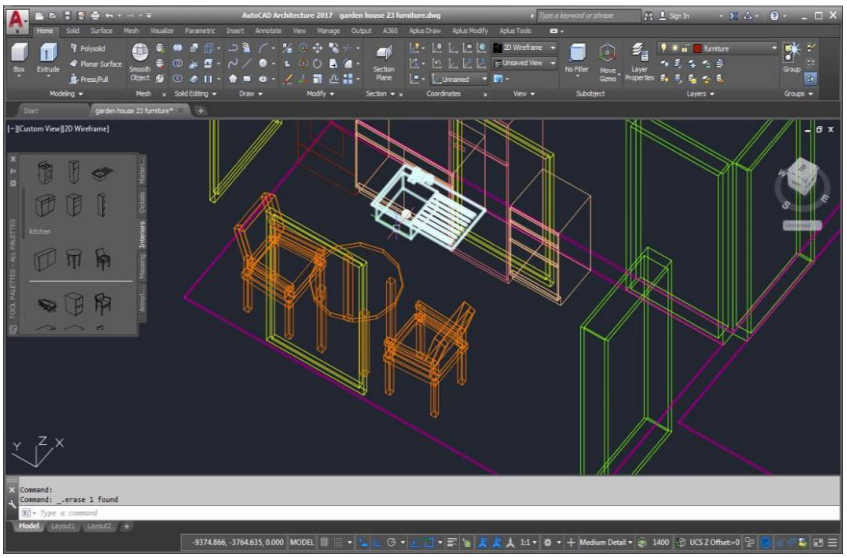

Рисунок 2.3 — Інтерфейс AutoCAD

Громіздкість цих систем залишається головним недоліком. Звичайному користувачеві буде тяжко орієнтуватися і неможливо автоматизувати потрібний процес(за винятком конструювання). Саме тому запропоновані дослідження є актуальними для вивчення.

**Висновок до розділу 2**

Таким чином, проаналізувавши існуючі програмні засоби до автоматизування процесів побудови та візуалізації приміщень у VR, було виділене наступне:

- Не існує ідеального аналогу запропонованої системи. Необхідний результат можливо досягнути лише шляхом використання декількох програмних продуктів.
- Програмні продукти, які мають схожий функціонал у побудові та завантаженні об’єктів, надто складні для звичайного користувача. Крім того, для використання побудованих моделей у прикладних задачах, необхідно застосовувати інструменти інших програмних комплексів.
- Аналоги, перечисленні вище, мають надзвичайно складний інтерфейс і громіздкий функціонал. Дані програмні реалізації націлені на професіоналів, тому звичайному користувачеві буде не зрозуміло як працювати в системі.

# 3. ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

Після детального аналізу поставленої задачі та методів її вирішення, було очевидно виявлено, що необхідно розроблювати програмний комплекс на основі платформи Unity. Головною перевагою цього рушія гри перед іншими варіантами є його універсальність, зручність і можливість використання на будь-яких пристроях. Це ідеальний «фундамент» для розробки запропонованої системи. За сценарну мову програмування, на якій пишеться ігрова логіка, було за замовчуванням обрано C#. За інструмент побудови VR візуалізації, в свою чергу, було обрано Google VR SDK.

## 3.1 Опис архітектури та платформи програмного комплексу

Для реалізації поставленої задачі було вирішено використовувати ігровий рушій Unity.

Ігровий рушій Unity – це неперевершена розширюваність і готовність до адаптації в стрімко мінливих умовах завдяки потужній системі скриптинга на C#, багатим API і документацією, який пропонує різні варіанти доступу до вихідного коду для розробки на C++[7]. Його основні функціональні можливості мають всі необхідні для розробки засоби. Редактор має зручний і простий Drag & Drop інтерфейс. Також цей ігровий рушій ділиться на сцени (рівні ігрової логіки) – окремі файли, які зберігають у собі свої ігрові світи зі своїм набором об'єктів і сценаріїв. Сцени, в свою чергу, можуть містити в собі об'єкти. Об'єкти – це ігрова одиниця, яка містять набори компонентів, з якими і взаємодіє написана логіка(скрипти).

Unity також підтримує фізику твердих тіл і тканин, що надає реалістичності віртуального світу.

Бізнес-логіку у реалізації програмного продукту можна описати: ігровим рушієм(те, що нам надає платформа Unity), логікою гри(частина роботи програміста) і ресурсами програмного забезпечення. Тут модель не відділена від уявлення – кожен об'єкт несе в собі як дані, так і засіб відображення. Таким чином,

можна зрозуміти, що немає необхідності у побудові архітектури так як Unity несе в собі зручну архітектурну реалізацію.

### 3.2 Скриптова система ігрового рушія

Скриптова система відіграє важливу роль у створенні будь-якого програмного продукту. Цей факт є очевидним через те, що скрипт, по своїй суті, є програмним файлом сценарію(поведінки системи), який автоматизує деяку задачу, яку користувач робив би вручну, використовуючи інтерфейс програми.

Поведінка об'єктів запропонованого ігрового рушія, в свою чергу, контролюється за допомогою компонентів (Components), які приєднуються до них. Незважаючи на те, що вбудовані компоненти Unity можуть бути дуже різнобічними, в ході розробки може стати зрозумілим, що вам цих можливостей недостатньо, щоб реалізувати ваші власні особливості геймплея. Щоб виправити це, Unity дозволяє створювати свої компоненти, використовуючи скриптову систему. Вона дозволяє активувати ігрові події, змінювати параметри компонентів, і відповідати на введення користувача яким завгодно способом.

Вже відомо, що скриптова система Unity платформи зроблена на Mono. Технологія Mono – це проект по створенню повноцінного втілення вже відомої системи .NET. Mono включає в себе компілятор мови програмування C#, середовище виконання .NET і ряд необхідних бібліотек, які не тільки дають змогу працювати напряму з Unity рушієм, але і реалізовувати необхідно логіку за межами ігрової платформи. Розробники програмного забезпечення можуть використовувати так званий UnityScript(власна скриптова мова) або Boo (мова програмування, схожа на Python). За замовчуванням, C# обраний за стандартну мову програмування.

.NET Framework – програмна платформа, яка була випущена компанією Microsoft в 2002 році. Основою цієї платформи є Common Language Runtime (CLR), яка підходить для різних мов програмування. Функціональні здібності CLR доступні в будь-яких мовах програмування, які використовують це середовище.

C# – мова програмування, яка поєднує об’єктно-орієнтовані та контекстно-орієновані концепції [2]. Створена в 1998-2001 роках компанією Microsoft як основної мови розробки додатків для платформи Microsoft .NET. Компілятор входить в стандартне встановлення самої .NET [2]. Ця мова програмування має сувору статичну типізацію, яка підтримує поліморфізм, перевантаження операторів, атрибути, події, властивості, винятки, коментарі[5].

Крім того, концепт скриптової системи полягає у наслідуванні класом створеним розробником від базового класу(MonoBehavior class), з якого проходить кожен сценарій. Життєвий цикл MonoBehavior складається з ряду функцій подій, які виконуються у заздалегідь визначеному порядку під час виконання сценарію(рисунок 3.1). Цей порядок виконання описано нижче:

— Awake (): Ця функція завжди викликається перед будь-якими функціями Start, а також відразу після а збірний інстанціюється.

— OnEnable (): Ця функція викликається відразу після ввімкнення об'єкта. Це відбувається, коли створюється екземпляр MonoBehaviour, наприклад, коли завантажується рівень або а GameObject зі сценарієм компонент інстанціюється.

— Start (): Запуск викликається перед першим оновленням кадру, лише якщо включений екземпляр сценарію.

— FixedUpdate (): FixedUpdate часто викликається частіше, ніж Оновити . Його можна назвати кілька разів на кадр

— Update (): оновлення викликається один раз на кадр. Це основна функція робочих коней для оновлення кадру.

— LateUpdate (): LateUpdate викликається один раз на кадр, після завершення оновлення . Будь-які обчислення, які виконуються в Оновлення, будуть завершені, коли розпочнеться LateUpdate .

— OnGUI (): Викликається кілька разів на кадр у відповідь на події GUI.

— OnApplicationQuit (): Ця функція викликається на всіх ігрових об'єктах до завершення програми. У редакторі його викликають, коли користувач зупиняє playmode.

— OnDisable (): Ця функція викликається, коли поведінка стає відключеною або неактивною.

— OnDestroy (): Ця функція викликається після всіх оновлень кадру для останнього кадру існування об'єкта (об’єкт може бути знищений у відповідь на Object.Destroy або при закритті сцени).

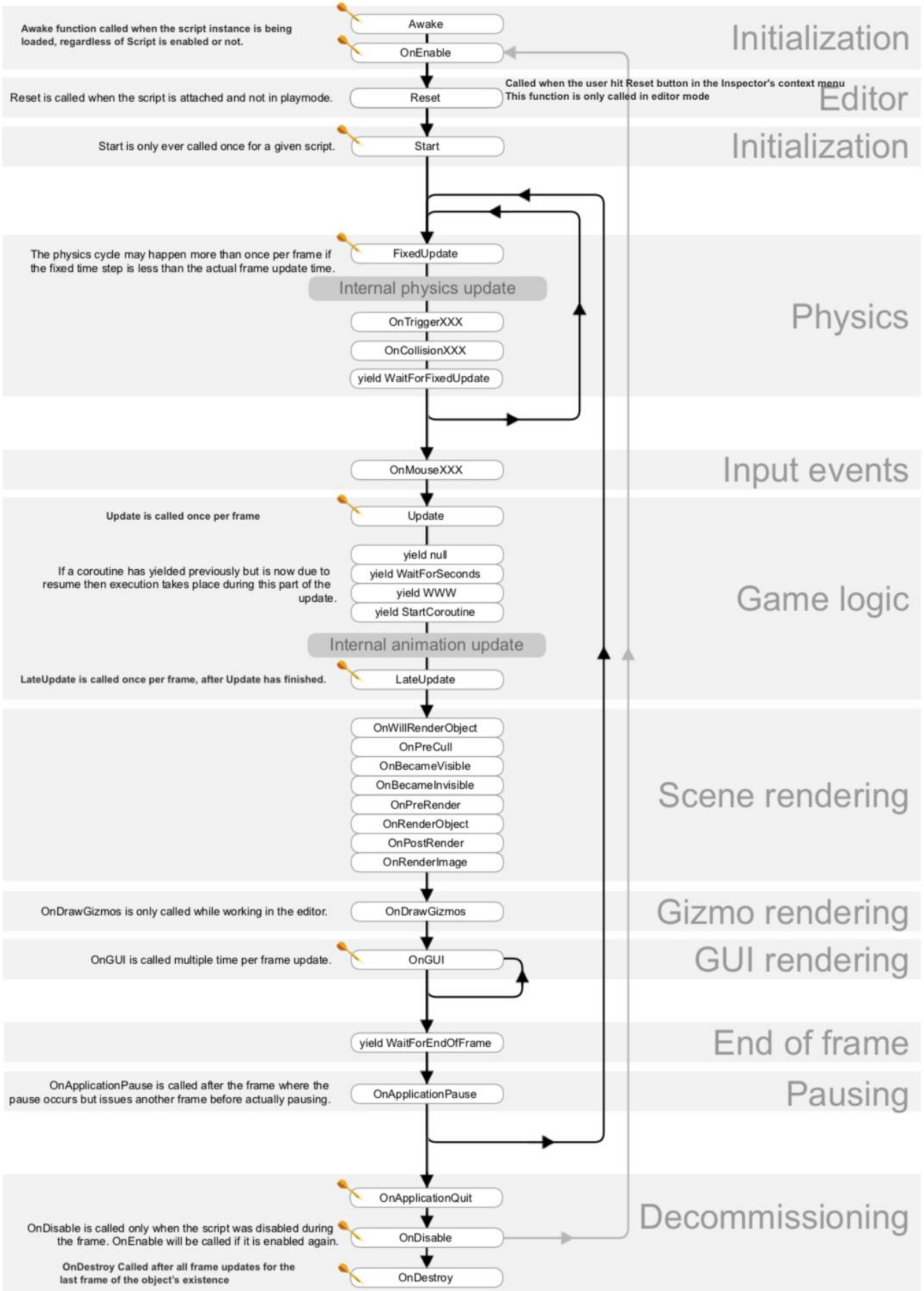


Рисунок 3.1 — Життєвий цикл Unity-Mono.

### 3.3 Інструменти штучного інтелекту ігрового рушія

Іноді, при розробці програмного продукту, потрібно, щоб персонажі(об’єкти) штучного інтелекту переміщалися в межах віртуального світу, слідуючи по встановленій або точно заданій траєкторії. Задачі, до яких застосовується ця технологія, можуть бути абсолютно різними: від простих задач NPC до складних агентів, які досліджують величезну ігрову мапу. Враховуючи обрану платформу і запропонований функціонал програмного продукту, було вирішено використовувати вбудований генератор навігаційних мешів Unity (NavMesh), який може сильно спростити пошук шляхів для агентів.

NavMesh – це така навігаційна сітка, яка є абстрактною структурою даних і використовується в різних додатках штучного інтелекту для орієнтування агентів на пошук шляхів від однієї точки до іншої[9]. Іншими словами, NavMesh робить так, що агент проходить через сцену, враховуючи при цьому всі межі поточної мапи, не проходячи через стіни і не «падаючи» в текстури.

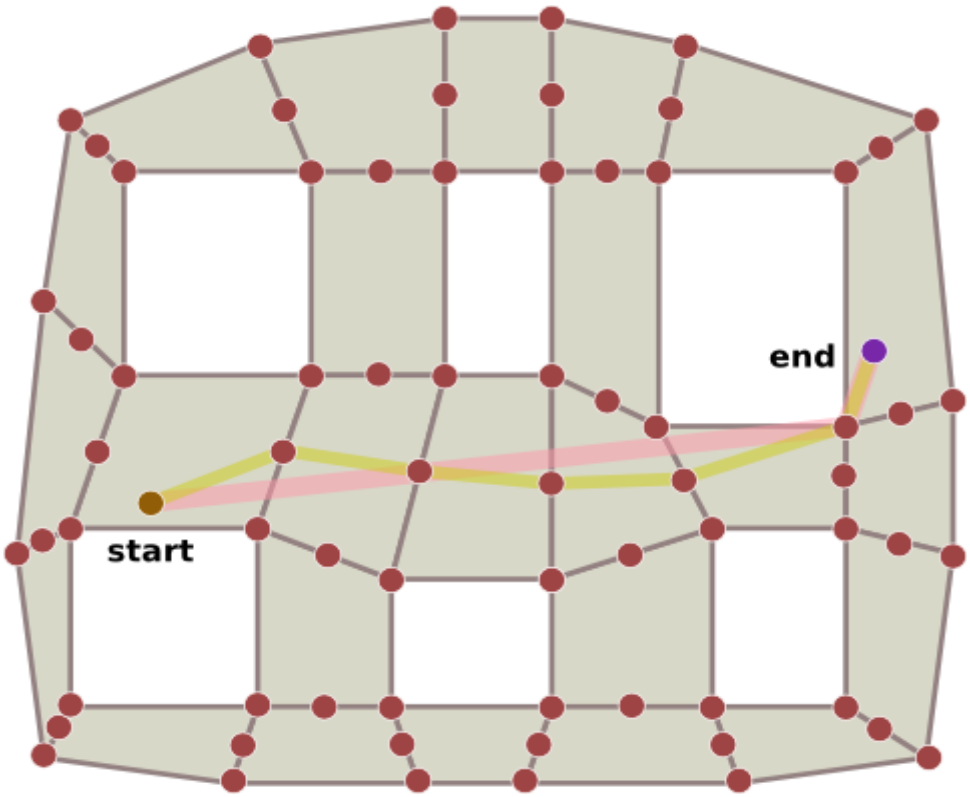


Рисунок 3.2 — Системи, яка використовує NavMesh для переміщення агента від початкової до кінцевої точки.

Варто зазначити, що основою штучного інтелекту є такі інструменти:

— Сітка: Це поле, яке складається з точок та ліній. У цьому контексті мається на увазі багатокутна сітка, яка утворена декількома лініями та багатокутниками. Таким чином, сцену можна розглядати як сітку.

— Агент: суб'єкт господарювання, здатний здійснювати діяльність сам. У даному випадку говориться про ворога чи NPC.

— Інструмент запікання(MeshBaker): головна функція, яка визначає точні межі мапи в залежності від завантаженої сітки[12].

Кожен інструмент є важливим і необхідним.

Крім того, слід розуміти, що в ході реалізації необхідного функціоналу штучного інтелекту ігрового рушія треба враховувати той факт, що агенти повинні вміти самостійно визначати маршрут, і якщо не можуть досягти потрібної точки, то вони повинні вміти прокладати найбільш ефективний маршрут і змінювати свій шлях, коли на шляху з'являються перешкоди(рисунок 3.3).

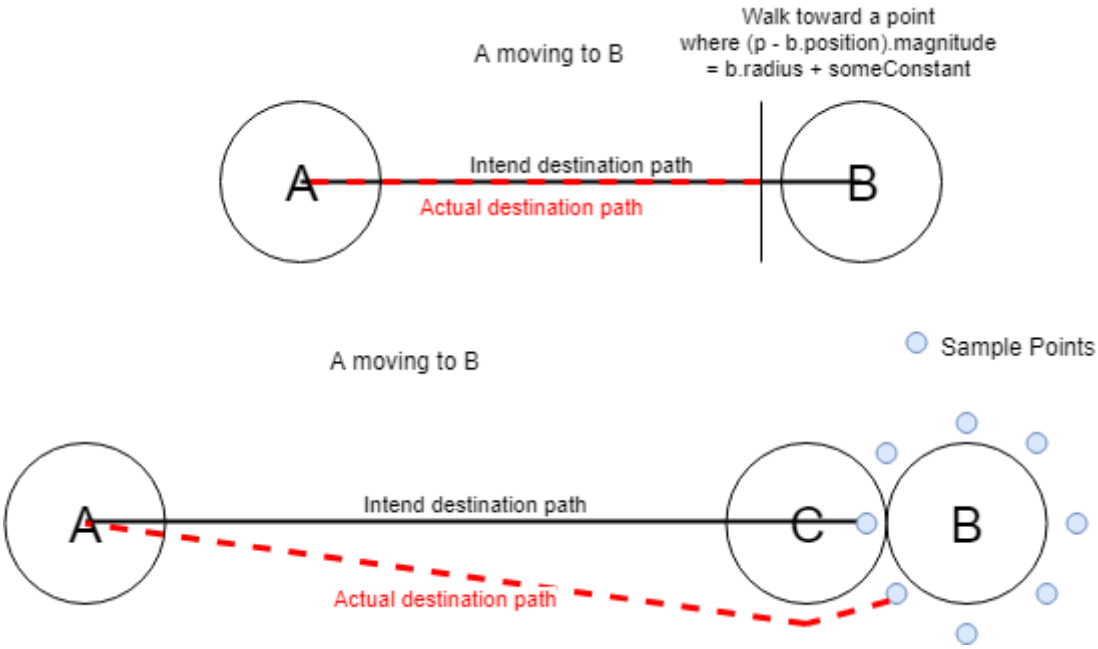


Рисунок 3.3 — Схема встановлення шляху штучним інтелектом.

### 3.4 Технологія віртуальної реальності

Для візуалізації автоматизованості і деякого функціоналу системи в більшості випадків використовують технологію віртуальної реальності.

Віртуальна реальність – світ, створений технічними засобами, який передається людині через його відчуття взаємодії зі світом: зором, слухом, дотиком та ін.[9]. Віртуальна реальність імітує як вплив, так і реакції на вплив. Для створення переконливого комплексу відчуттів реальності, комп'ютерний синтез властивостей і реакцій віртуальної реальності проводиться в реальному часі. Для розробки цієї самої імітації в Unity, необхідні спеціальні інструменти – пакети. Рисунок 3.4 наглядно демонструє концепт імітації віртуальної реальності.

Рисунок 3.4 — Імітація віртуальної реальності.

Пакет Google VR ідеально підходить для таких задач, адже підтримує створення програм для пристроїв, що використовують Daydream / Cardboard імітатори. Пакет підтримується через технології нативної віртуальної реальності в Unity. Він дозволяє створювати вражаючі кросплатформові VR-ефекти. Завдяки важливим функціям VR, таким як відстеження руху, стереоскопічна візуалізація та

взаємодія з користувачем, можна створювати абсолютно сучасні враження від VR або покращити уже наявні додатки, що підтримують VR.

На сьогоднішній день існує новий альтернативний пакет від Unity, Unity XR Tech Stack. Проте, доступний він лише на платній основі.

### 3.5 Особливості підготовки сцени до генерації у VR

Конструювання приміщення – етап підготовки до генерування, за допомогою якого користувач може побудувати власну модель (яка необхідна для логіки візуалізації).

Аналізуючи обрані програмні реалізації, було досліджено декілька шляхів вирішення цієї проблеми: маніпулюючи мешами в реальному часі або ж за допомогою динамічного створення префабів по точкам.

Обидва методи вирішення цією задачі дадуть необхідний результат. Проте, специфіка та сфера застосування у них абсолютно різна.

Маніпуляція мешами в реальному часі. При стандартному підході в Unity для рендеринга моделі використовуються компоненти MeshFilter і MeshRenderer . MeshFilter посилається на Mesh - Ассет, який представляє модель.(рисунок 3.5)


Рисунок 3.5 — Компонент MeshFilter і MeshRenderer в інспекторі UnityEditor.

У Unity і в багатьох інших ігрових рушіях використовується популярний спосіб опису моделей: за допомогою масивів вершин, трикутників і нормалей. Додатково для текстурування використовуються uv-координати вершин(рисунок 3.6). Для роботи з моделями є клас Mesh, в якому для кожного набору даних є окремий масив. У Mesh.vertices зберігаються координати вершин, в Mesh.triangles - індекси вершин групами по три. А в Mesh.normals і Mesh.uv лежать вектори нормалей і координати uv-карт, індекси яких повинні збігатися з індексами відповідних вершин. Працюючи з цими даними, можна будувати та маніпулювати об’єктами в реальному часі. Проблема застосування цього методу- по яким значенням буде будуватися модель, адже від цього буде залежати точність побудованого приміщення. Таким чином, цей метод конструювання підходить для вирішення необхідних задач, але не є оптимальним.

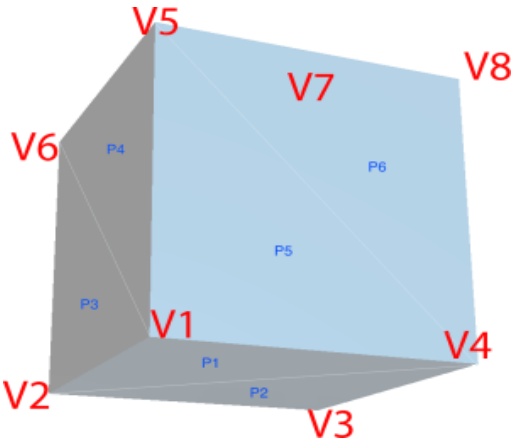


Рисунок 3.6 — Меш куба, що містить вершини та багатокутник.

Динамічне створення префабів по точкам. Основа концепту префабів (Prefabs) полягає у створенні єдиної моделі для подальшого використання(рисунок 3.7). Префаб – це набір заздалегідь встановлених ігрових об'єктів(GameObjects) і компонентів(Components), які використовуються більше одного разу за всю «гру». Префаби припадають дуже до речі, коли необхідно створити екземпляри складних ігрових об'єктів під час процесу взаємодії користувача з системою.

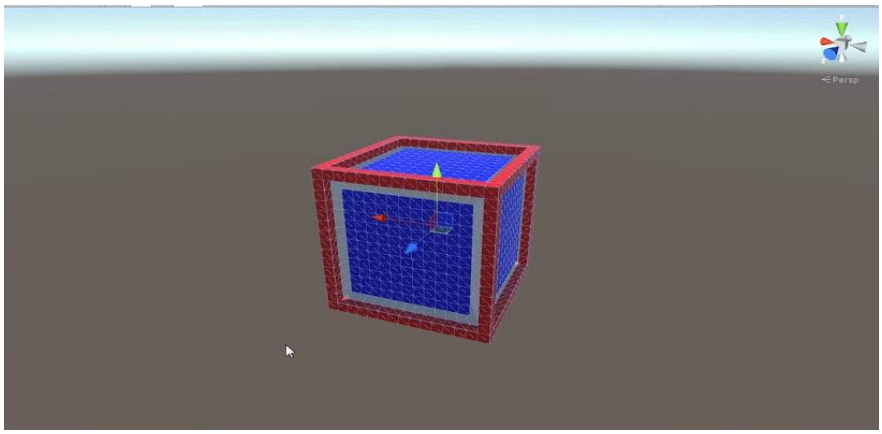

Рисунок 3.7 — Процес створення префабу.

У даному методі головна задача полягає у зчитуванні позиції вводу користувача, і на основі цих даних побудувати необхідну модель(стіну, двері, вікна та інші).

Таким чином, створення екземплярів префабу має багато переваг над іншими підходами конструювання і підготовки приміщення до генерування VR-сцен:

- Легше будувати приміщення через код.
- Результат набагато точніший за аналогів. Адже приміщення будується по префабам які мають відомі розміри.
- При створенні VR-сцени, системі легше створити маршрут для штучного інтелекту(агента).

### 3.6 Завантаження готової моделі в межах реального часу

У ході проектування системи, було досліджено два метода реалізації цієї задачі: завантаження та обробка зображення 2-д плану та завантаження готової 3-д моделі.

Під час створення системи, яка може завантажувати зображення 2-д плану(рисунок 3.8) та оброблювати дані попіксельно, було виявлено, що даний метод реалізації є неточним і має великі похибки у побудові. В залежності від розміру зображення кількість пікселів може варіюватися. Розмір елементів

приміщення повністю залежать від кількість оброблених пікселів – отже приміщення не може бути побудоване відповідно до реальних масштабів.

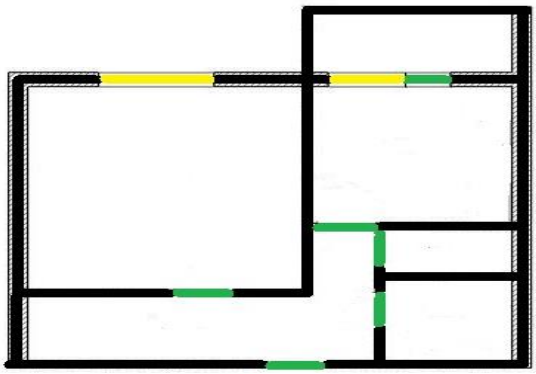

Рисунок 3.8 — Зображення на основі якого було проведено дослідження.

Завантаження готової 3-д моделі, в свою чергу, стало ідеальним варіантом для вирішення даної задачі. В основі концепту цього методу лежить створення Ассет папки, яка доступна для проекту без його завантаження в якості частини сцени. Наприклад, вона може містити персонаж або інший об'єкт, який може з'явитися в будь-якому місці сцени, але який буде використовуватися лише зрідка (це може бути "секретна" функція, повідомлення про помилку або попередження про рекордний рахунок). Крім того, існує можливість завантаження ресурсів з окремого файлу або завантаження об’єктів через URL у дану папку, щоб скоротити час початкового завантаження або додати в проект взаємозамінний контент через код у режимі реального часу.

Завдяки функціоналу ігрового рушія(Asset Bundles) і мові програмування C#, яка підтримує потоки файлової системи(рисунку 3.9, зображення місця класу файлових потоків у ієрархії класів простору System.IO), реалізація завантаження моделей приміщення, які були експортовані з інших програм не складатиме проблем.

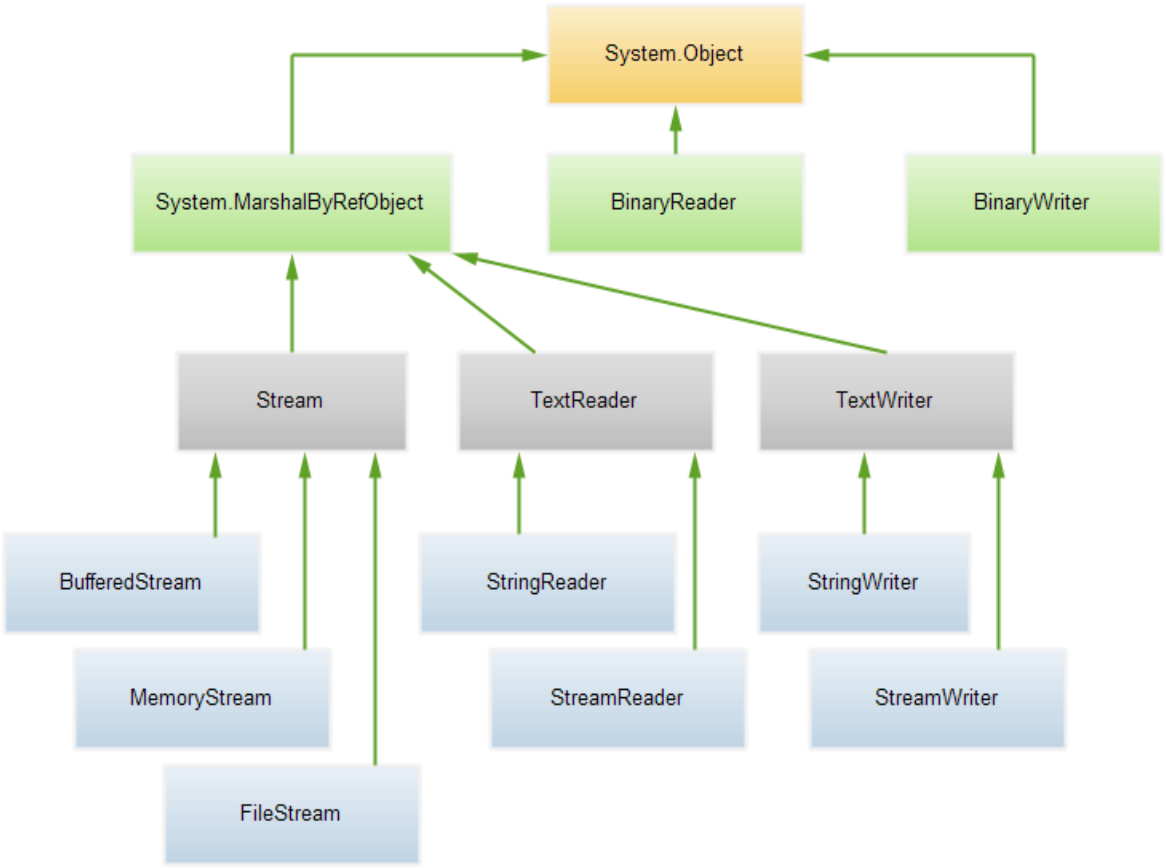


Рисунок 3.9 — Ієрархія класів в просторі імен System.IO

.

### 3.7 Обмін даних з іншими програмними комплексами

Часто буває, що при розробці того чи іншого програмного продукту, необхідно реалізувати обмін даних між системою та чужим програмним комплексом.

JSON (JavaScript Object Notation) - це такий легкий формат обміну даними, який заснований на підмножині стандарту мови програмування. Він повністю незалежний від мови, але використовує відомі конвенції, знайомі програмістам сімейства мов C. Саме ці властивості роблять JSON ідеальною мовою для обміну даними між додатками та серверами.

В основі JSON лежить дві конструкції:

• Колекція пар імен / значень. У різних мовах це реалізується як об'єкт , запис, структура, словник, хеш-таблиця, список клавіш або асоціативний масив.

• Впорядкований список значень. У більшості мов це реалізується як масив , вектор, список чи послідовність[5].

Варто зазначити, що практично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Зрозуміло, що формат даних, який взаємозамінний з мовами програмування, також повинен базуватися на цих структурах. В межах системи JSON вони набувають такої форми:

• Об'єкт є неврегульованим набором пар ім'я / значення(рисунок 3.10).

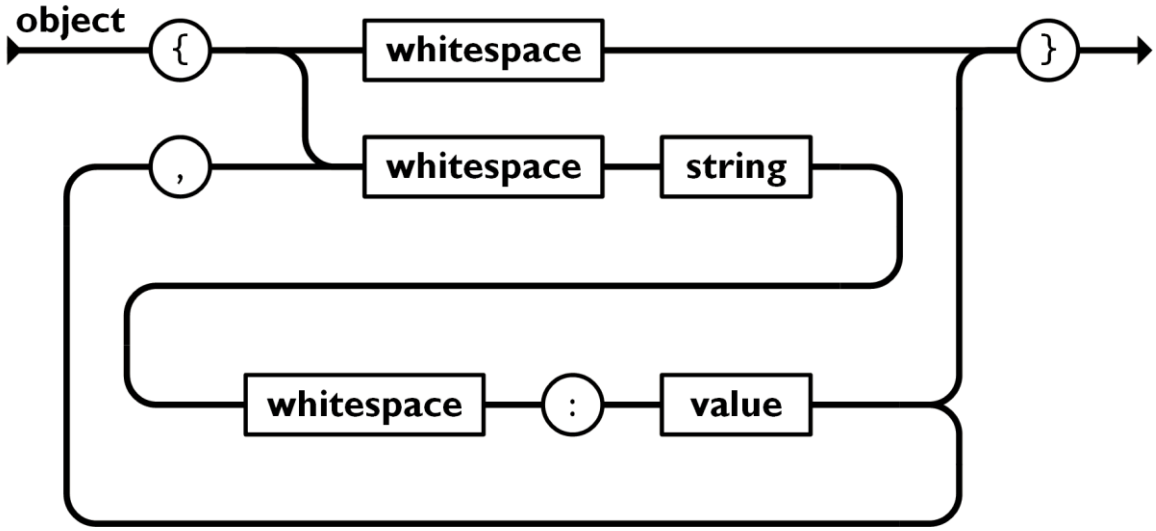


Рисунок 3.10 — Форма об’єкту JSON.

На даному рисунку представлена форма об’єкту у мові обміну даними. Як ми бачимо, Об'єкт починається з лівої дужки «{» і закінчується правою дужкою «}». Кожне ім'я супроводжується двокрапкою і парою ім'я / значення, відокремлені один від одного комою.

• Масив являє собою упорядковану сукупність значень(рисунок 3.11).

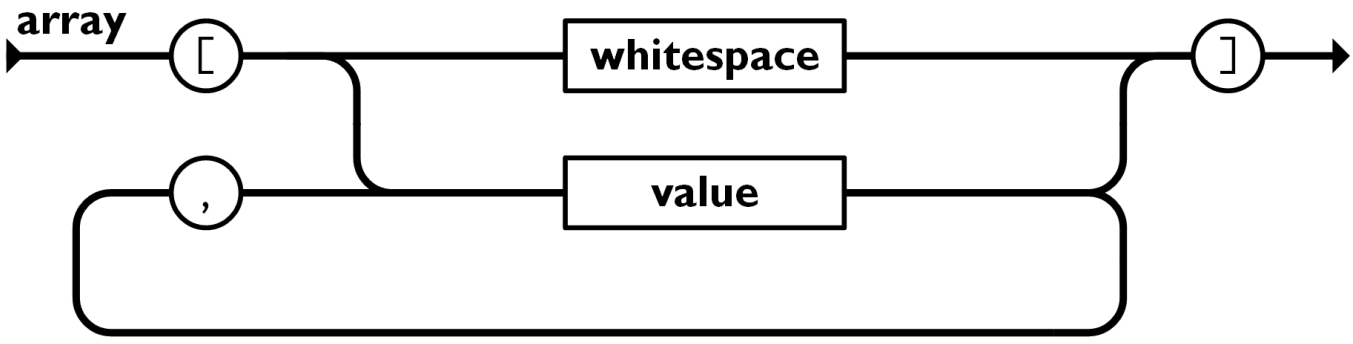


Рисунок 3.11 — Форма масиву JSON.

Варто зазначити, що масив лежить в межах дужок «[]». Значення відокремлені один від одного комами.

• Значення ж, може бути числовим або буквеним, істинним, помилковим або нульовим, чи мати форму об'єкта або масива(рисунок 3.12) . Ці структури можуть вкладатись.

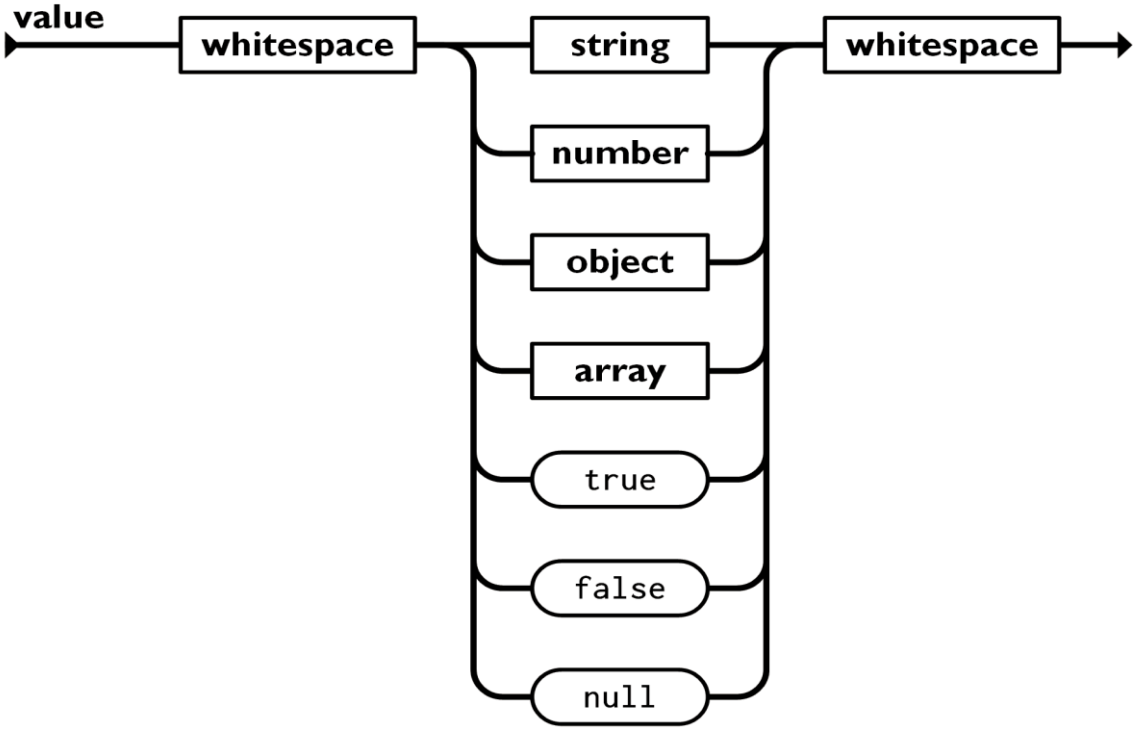


Рисунок 3.12 — Форма значення JSON.

• Рядок являє собою послідовність віл нуля та більше символів Unicode, загорнутих в подвійних лапках або зворотніх слешах(рисунок 3.13).

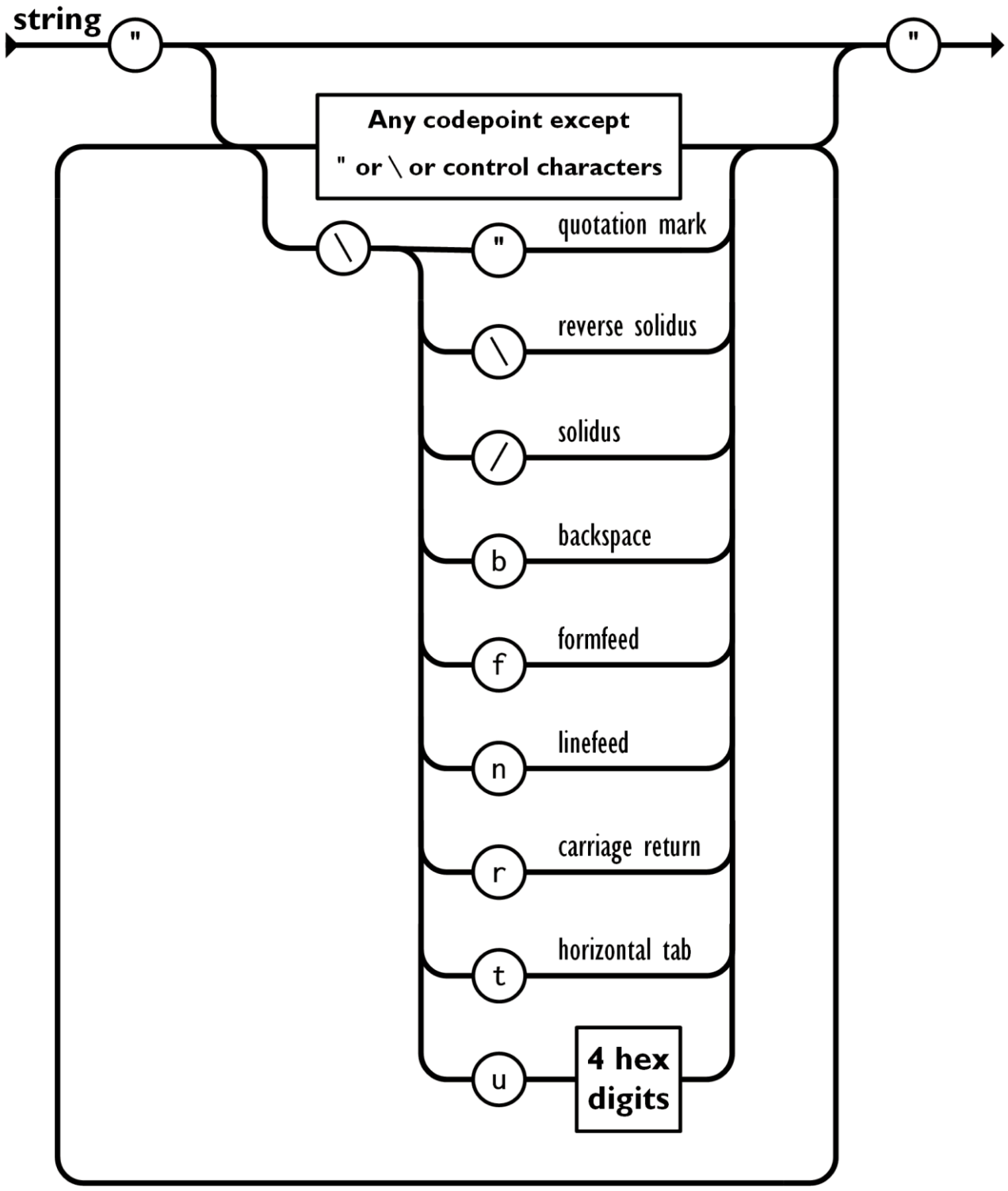


Рисунок 3.13 — Форма рядка JSON.

Крім того, String мови програмування C або Java мають дуже схожу структуру.

- Число дуже схоже на ряд C або Java, за винятком того, що не використовуються восьмеричні і шістнадцяткові формати(рисунок 3.14).

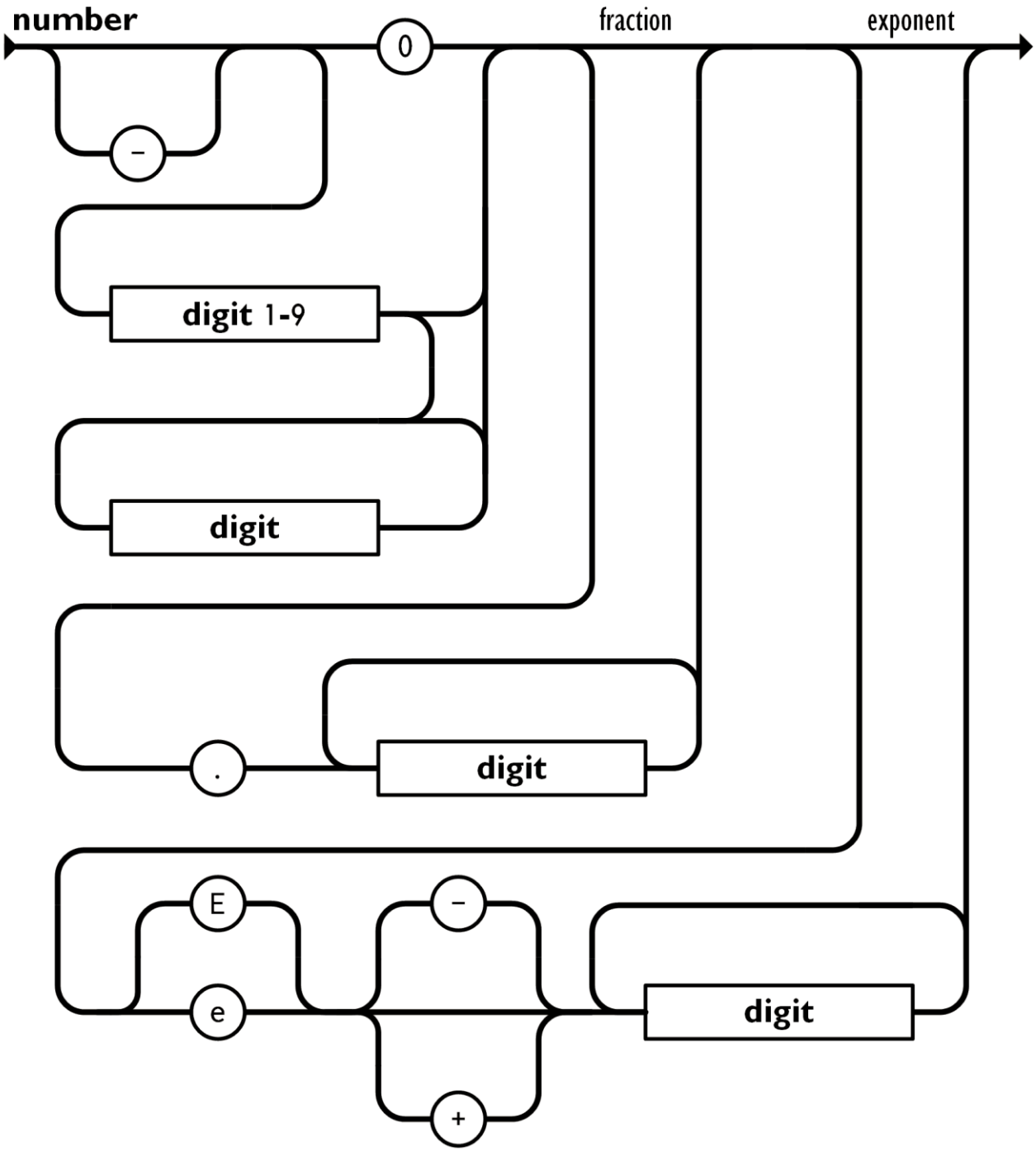


Рисунок 3.14 — Форма числа JSON.

Пробіл можна вставити між будь-якою парою лексем(Рисунок 3.15). За винятком кількох деталей кодування, які повністю описують мову.

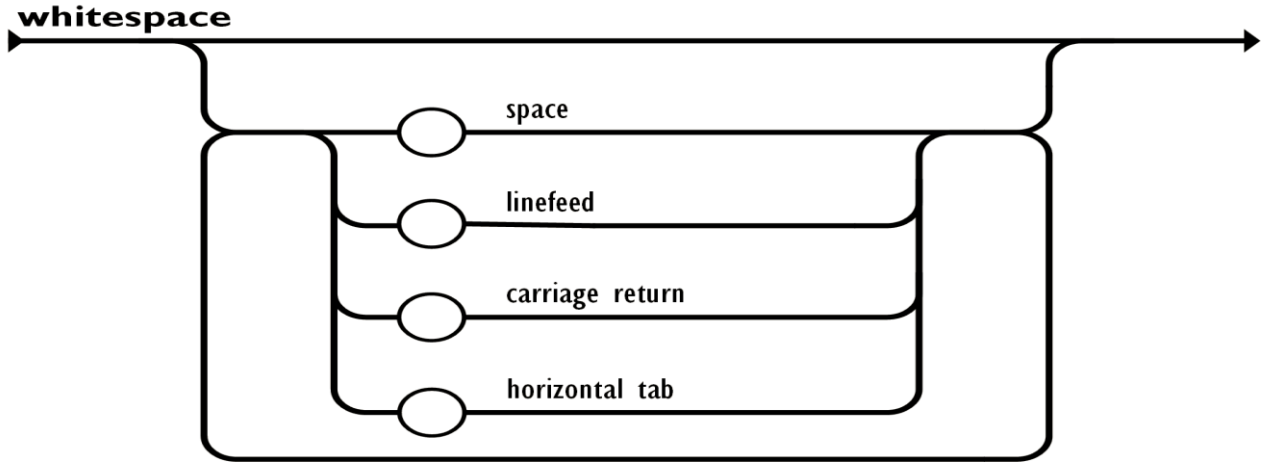


Рисунок 3.15 — Форма пробілу JSON.

Для оброки цієї інформації, починаючи з версії 5.3.3, Unity додала JsonUtility до свого API. Тому, можна забути про різні сторонні бібліотеки, хіба що якщо у розробці щось дуже складне.

JsonUtility швидше, ніж сторонні бібліотеки, а й, між тим, підтримує тільки прості типи: не підтримує колекції, такі як словник (Dictionary). Винятком є список (List) і масив списків (List array). Знання синтаксичної структури файлів формату JSON легко дозволить отримати та обробити всі необхідні дані.

## 3.8 Остаточне обґрунтування вибору програмних застосунків та методів реалізації

Проектуючи запропоновану систему, було проаналізовано та вивчено предметну область.

Зрозумівши вище написаний опис платформи, можна зрозуміти, що Unity – більше, ніж ігровий рушій, це середовище для розробки багатоплатформових ігор і додатків, в якій об'єднані різні програмні засоби, що використовуються при створенні ПЗ – текстовий редактор, компілятор і так далі. При цьому, завдяки зручності використання, Unity робить створення додатків максимально комфортним і зручним, а багатоплатформність двигуна дозволяє розробникам охопити якомога більшу кількість ігрових платформ і операційних систем.

Тому, на основі саме цих спостережень було вирішено розроблювати програмний продукт на технологіях ігрового рушія саме від Unity.

Технології, які використовуються для реалізації логіки програмного продукту, були обрані за принципом зручності у використанні, відкритості вихідних кодів з використанням обраної платформи, актуальності в наш час. Зокрема мова C#, яка була використана для написання необхідної логіки, ідеально підходить для задач які компілюються на проекті Mono.

Для візуального проектування програмного продукту у VR було обрано Google VR SDK. Цей вибір обумовлений дуже простим причинам: доступність у використанні, цей програмний пакет є повністю безкоштовним, цей програмний пакет має щільний зв’язок з Unity платформою.

Таким чином, дані технології в сукупності дають змогу створити якісний продукт, який зробить можливим використання необхідного функціоналу.

**Висновок до розділу 3**

Отже, досліджуючи та аналізуючи предметну область, а також вимоги, які були висунуті при проектуванні, було обґрунтовано вибір засобів реалізації запропонованої системи. Було визначено набір задач і методів, їх вирішення для побудови системи, а також проведено ряд експериментів і досліджень на кожному етапі проектування.

# 4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Система конструювання сцени, генерування її у VR і візуалізація сліпих зон реалізована на основі моделювання цього програмного забезпечення. Адже саме аналіз моделі дає повний, точний та адекватний опис запропонованої системи, що має конкретне призначення. В даному випадку, для запропонованої системи та її бізнес-процесів, використовується універсальна мова візуального моделювання(UML).

## 4.1 Функціональність системи

Для того щоб описати функціональну реалізацію запропонованої системи, була побудована діаграма прецедентів системи. Програмний застосунок для генерування і візуалізації сцени у VR містить у собі одного головного актора – потенційного користувача системи, та декілька функціональних сцен.

Функціональні сцени складаються з послідовних етапів, яким слідує користувач під час роботи з системою.

Першим етапом початку роботи користувача з системою є головне меню, де відбувається налаштування головних функцій. Головне меню є втіленням центральної бази, від якої залежить функціонал подальших сцен, а саме сцени-конструктора.

На рисунку 4.1 представлена вище вказана USE-CASE діаграма головного меню, яка описує функції та дії користувача у головному меню. На даному рисунку представлено те, що функціонал головного меню є досить простим. Користувач може продовжити проект на якому зупинився, створити новий проект, чи завантажити старий проект. Крім того, з головного меню можна налаштовувати фунціонал конструктора побудови приміщення. Користувач може налаштувати розмір стін, дверей та вікон.


Рисунок 4.1 — Діаграма прецедентів головного меню

Наступний етап роботи користувача з системою є головна сцена, де відбувається конструювання або завантаження 3D плану приміщення. В цій сцені користувач може застосовувати головний функціонал програми: самостійно конструювати приміщення, завантажити модель приміщення, завантажувати визначені в іншій програмі камери, досліджувати сцену, згенеровувати VR сцену.

На рисунку 4.2 представлена USE-CASE діаграма головної сцени, яка описує функції та дії користувача у головній сцені. На даному рисунку представлено які сегменти користувач може будувати в сцені та можливості, які доступні на ній.

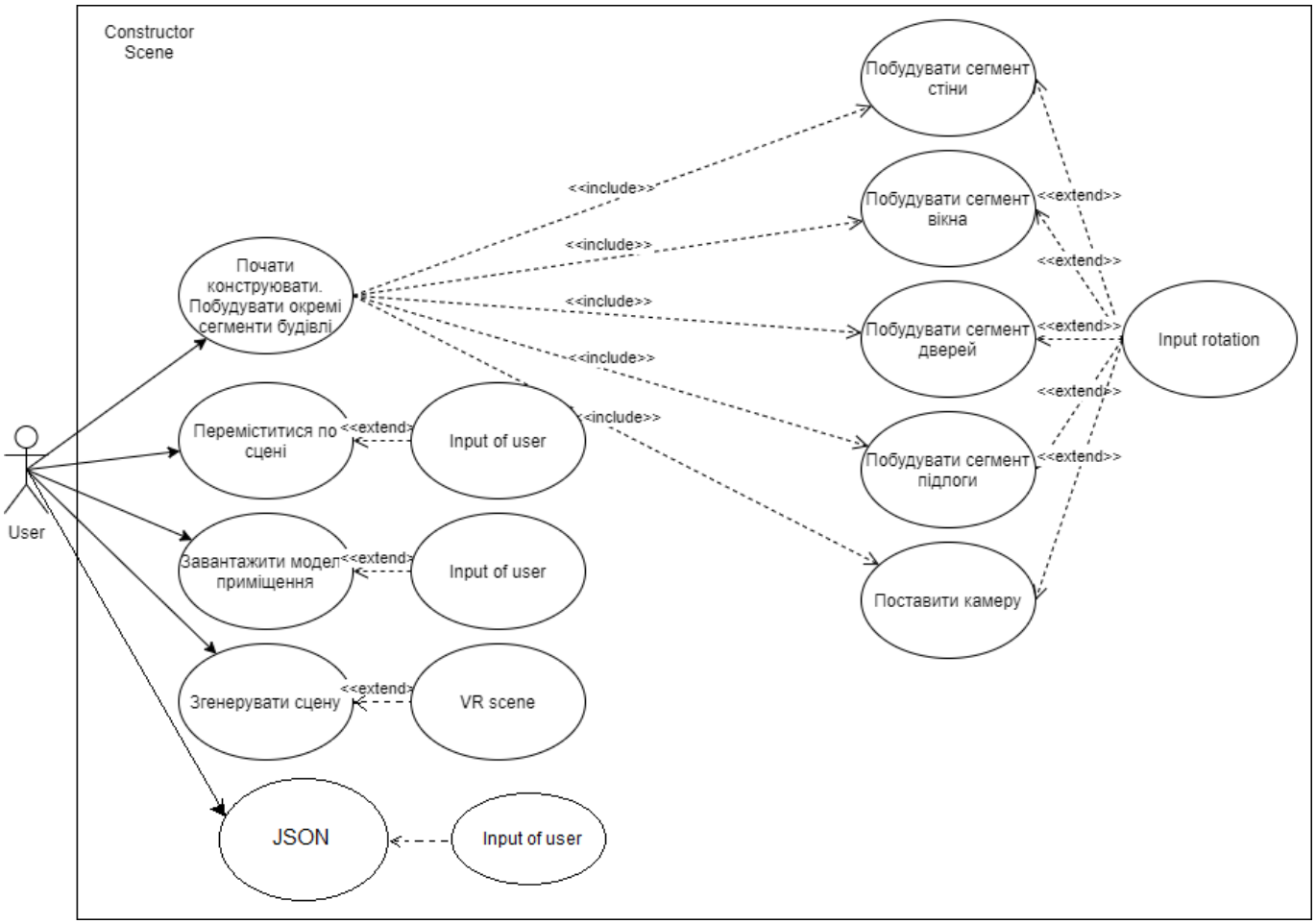


Рисунок 4.2 — Діаграма прецедентів головної сцени (Сцени-конструктора)

Завершальним етапом системи є взаємодія з згенерованою VR сценою. В даній сцені можна отримати лише візуальний результат, тому що користувач отримує приміщення з запущеними в неї агентами (штучним інтелектом, який досліджує будівлю), а сам знаходиться в режимі очікування. Камери, які були виставлені раніше, виступають для користувача засобом для слідкування.

### 4.2 Концептуальна модель даних

Так як система використовується лише локально(без доступу до інтернету), необхідність у серверній базі даних відпадає. Вся інформація про сцену і конструювання зберігається у користувача локально.

Таким чином, концептуальна модель збережених локально даних приведена на рисунку 4.3.

На даному рисунку видно, що будуючи чи завантажуючи 3D план приміщення, ці дані зберігаються локально у користувача.


Рисунок 4.3 — Концептуальна модель збережених локально даних

### 4.3 Діаграма класів системи

Діаграма класів – це основна логічна модель системи, що проектується. Вона призначена для представлення моделі статичної структури програмної системи в термінології класів об’єктно-орієнтованого програмування. Крім того, діаграма класів представляє собою граф, вершинами чи вузлами якого є елементи типу «класифікатор», які зв’язані різними типами структурних відношень

Класифікатор – спеціальне поняття, призначене для класифікації екземплярів, які мають загальні характеристики.

Таким чином, діаграма класів (рисунок 4.4) демонструє логічну модель системи, що проектується. Вона показує, що кожен не абстрактний клас наслідує MonoBehavior клас (клас, що демонструє поведінку об’єктів сцени ігрового рушія). Крім того, запропонована діаграма класів зображує всі існуючі класи та їх зв’язки:

— CreateSegments class – клас, який реалізує конструктор побудови приміщення в сцені 2.

— MousePosition class – клас, який реалізує пошук курсора в сцені 2.

— GlobalDataBase abstract class – абстрактний клас, який зберігає всі необхідні дані для генерації VR сцени (сцена 3).

— NavMeshBaker class – клас, який реалізує «запікання» приміщення для NavMesh агента (штучного інтелекту, який досліджує приміщення). Цей клас повністю залежить від того, яке приміщення було згенеровано раніше.

— GenerateSceneOnClickClass – це клас, який відповідає за генерування VR сцени.

— WalkingTarget class – це клас, який реалізує навігацію агента по сцені.

— AgentManager class – це клас, який реалізує створення агентів, їх початкову координату руху.

— VizualizationVR class – це клас, який реалізує VR візуалізацію для користувача.

— CameraEx abstract class – це клас, який реалізує навігацію користувача по сцені. Цей клас звертається до абстрактного класуVizualizationVR для того, щоб дізнатися, яка камера в даний момент бачить агента-ціль, і привести її в дію.

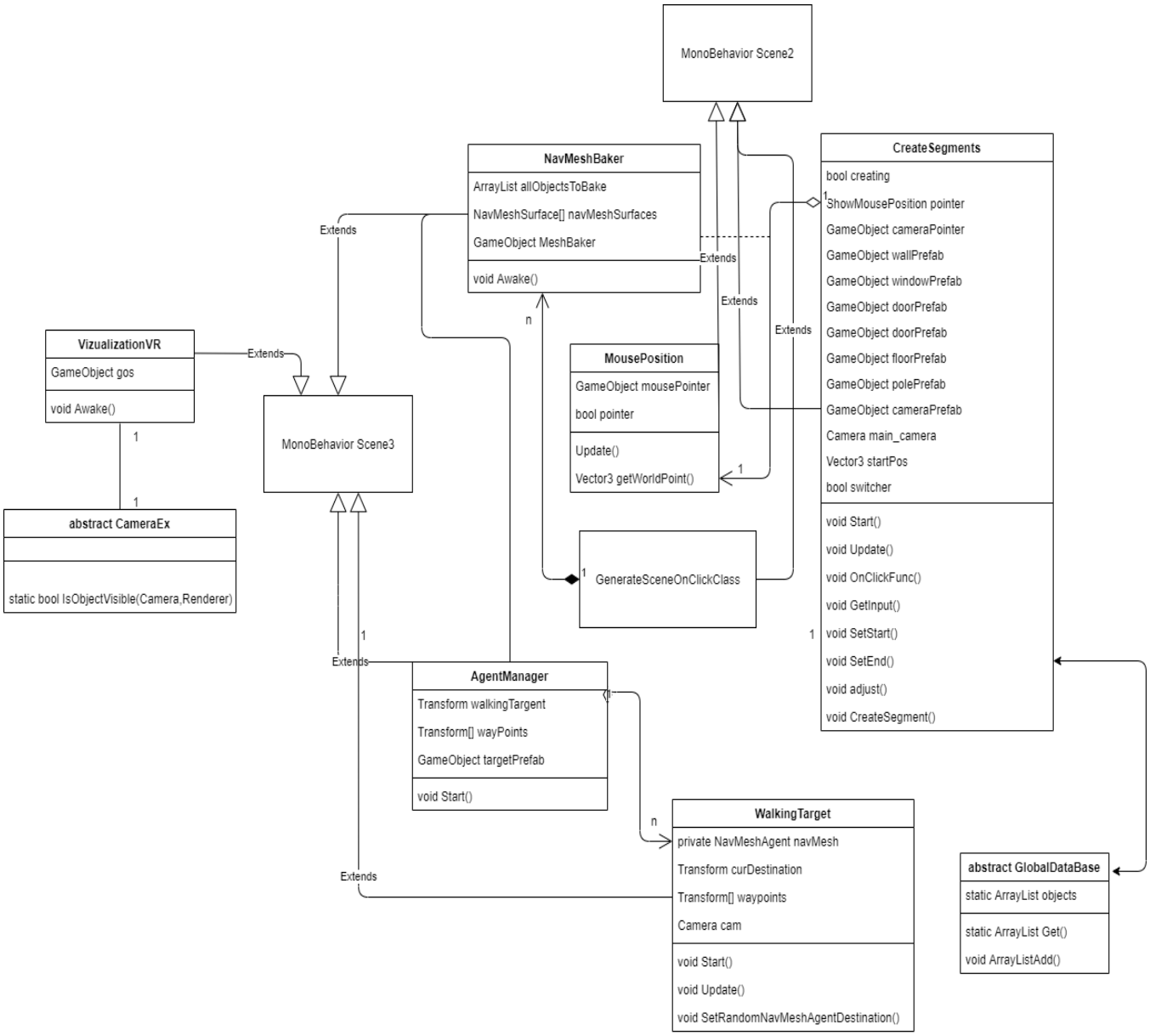


Рисунок 4.4 — Діаграма класів системи

## 4.4 Діаграма послідовності системи

Діаграма нижче (рисунок 4.5) відображає процедуру старту програми. Особливу увагу можна приділити хіба що альтернативному виходу зі сценарію і переходу до завантаження проекту, якщо є збережена незавершена партія.

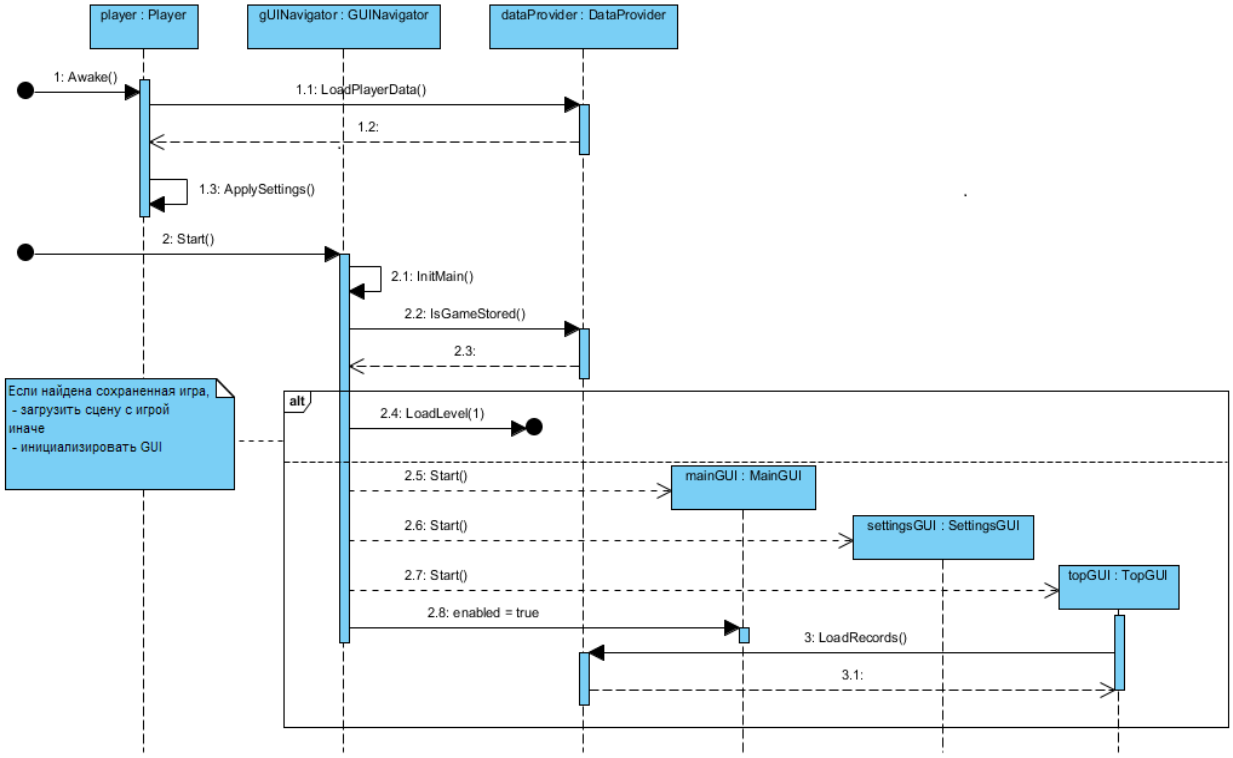


Рисунок 4.5 — Діаграма послідовності головного меню

Завантаження програми складається з обробки двох подій: awake і start. Подія awake обробляє об'єкт player: в цей момент він звертається до DataProvider – у для завантаження інформації про гравця, а потім викликає власний метод, який відповідає за застосування поточних налаштувань, наприклад, відключення звуку.

Подія Start обробляється трохи складніше:

- GUInavigator в методі initMain ініціює всі необхідні в цій сцені інтерфейси, кожному проставляючи ознака неактивності.
- Далі він перевіряє, чи не було збережено незавершеної гри. Це можливо, наприклад, при перериванні гри вхідним дзвінком.
- Якщо проект є, то відбувається завантаження ігрової сцени, а даний сценарій переривається.
- Умовно одночасно з цим обробляються події Start для всіх ініційованих класів інтерфейсів – в них відбувається додаткова ініціалізація, властива кожному конкретному інтерфейсу окремо (в т.ч. інтерфейс рекордів завантажує дані по таблиці рекордів, використовуючи dataProvider).

• Інтерфейсу головного меню присвоюється ознака активного, і він відображається користувачеві.

Далі розглянемо діаграму послідовності, яка ілюструє дії при запуску проекту. Точка початку – завантаження сцени з конструктором.

На даній діаграмі варто окремо відзначити альтернативний сценарій завантаження збереженої сцени, проте ми розглянемо систему роботи з пустою сценою (рисунок 4.6).

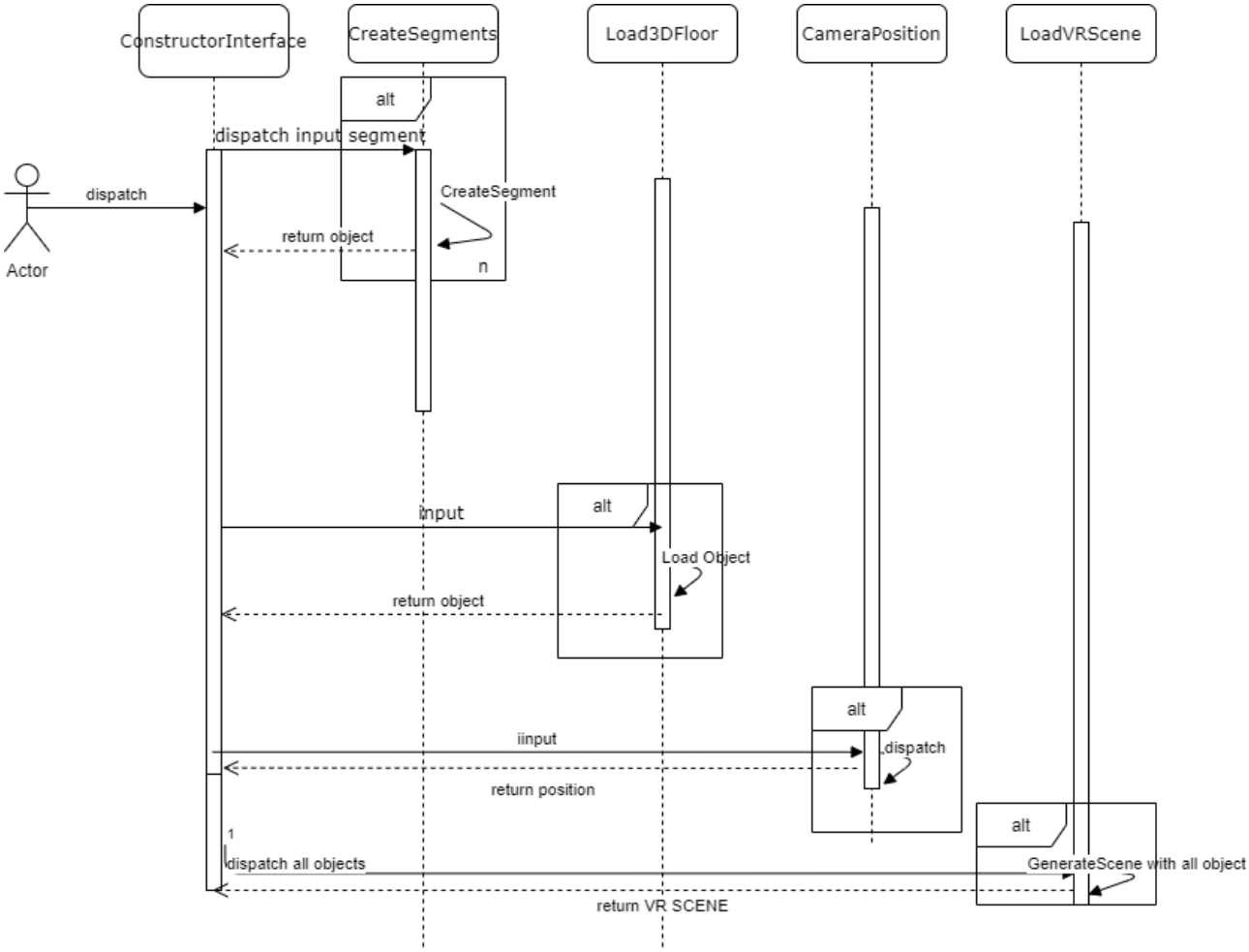


Рисунок 4.6 — Діаграма послідовності головної сцени

Далі система генерує VR-сцену і запускає агентів, що дає можливість користувачу виявити сліпі зони приміщення. Так як користувач є лише спостерігачем, немає необхідності створювати діаграми послідовності.

### 4.5 Розробка алгоритму конструювання приміщення

Алгоритм конструювання приміщення оснований на розміщенні мешу префабів та зчитуванні вводу користувача.

Для початку роботи алгоритму було створено префаби необхідних елементів приміщення(рисунок 4.13):

- Стіни(рисунок 4.7)

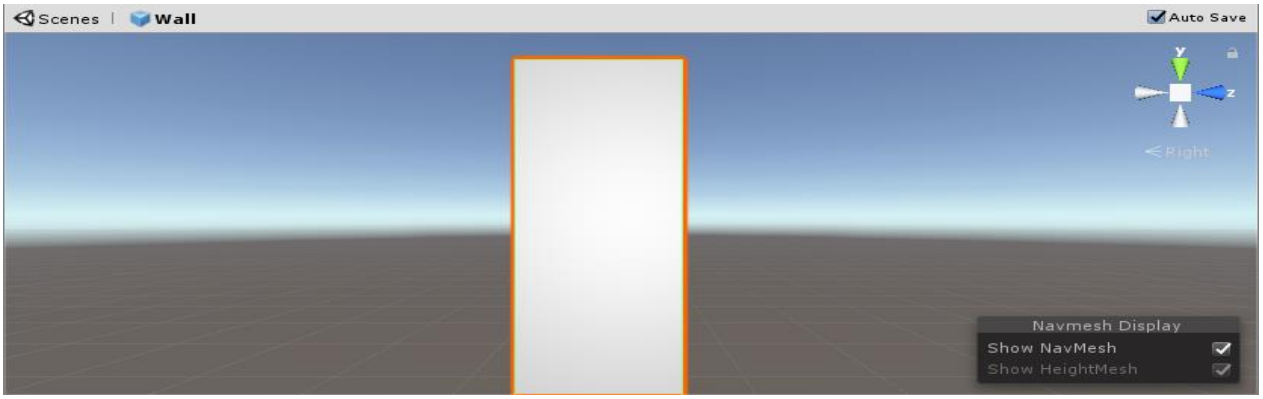


Рисунок 4.7 — Префаб стіни

- Дверей(рисунок 4.8)

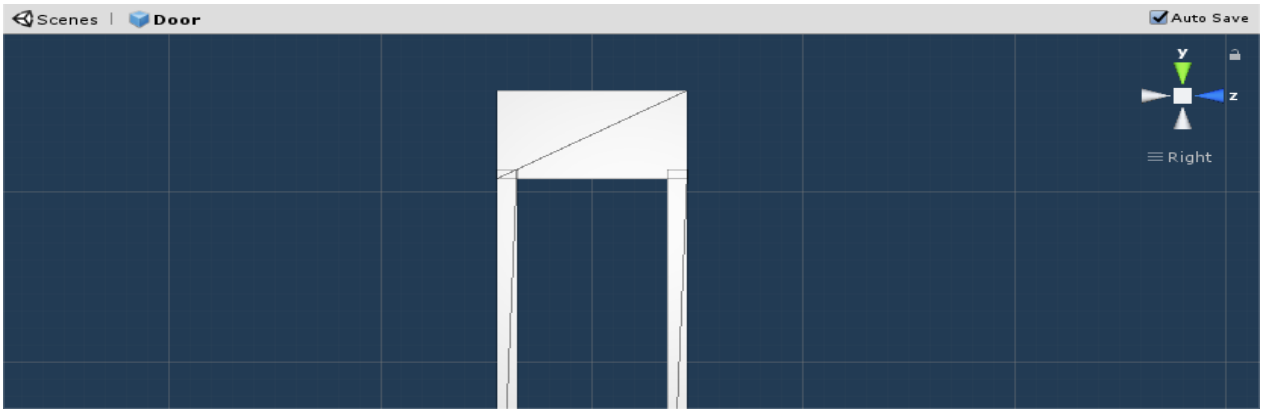


Рисунок 4.8 — Префаб дверей

- Вікон(рисунок 4.9)

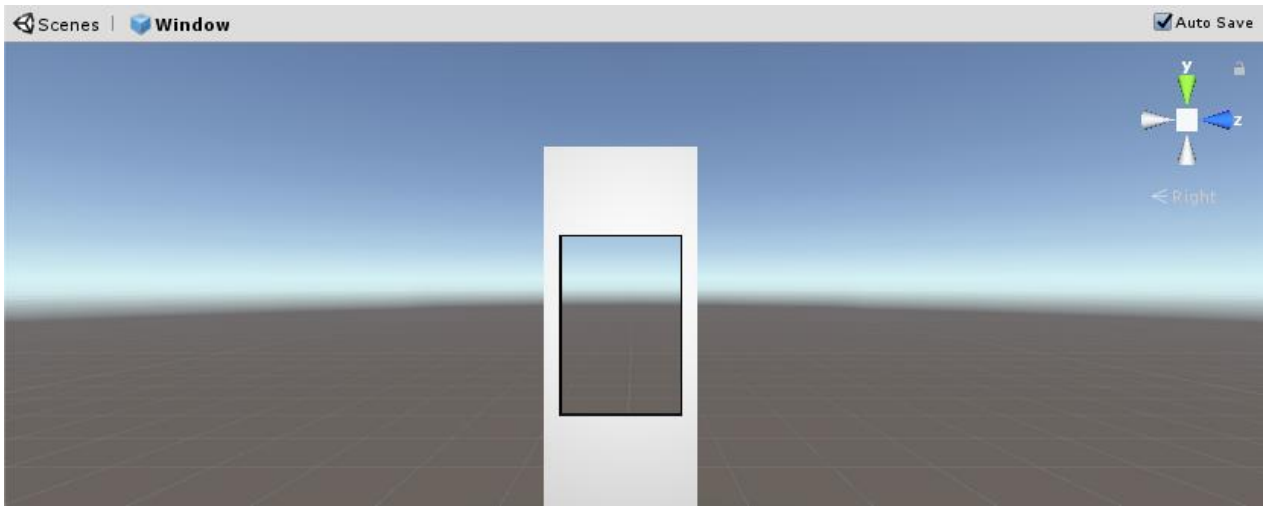


Рисунок 4.9 — Префаб вікон

- Підлоги(рисунок 4.10)

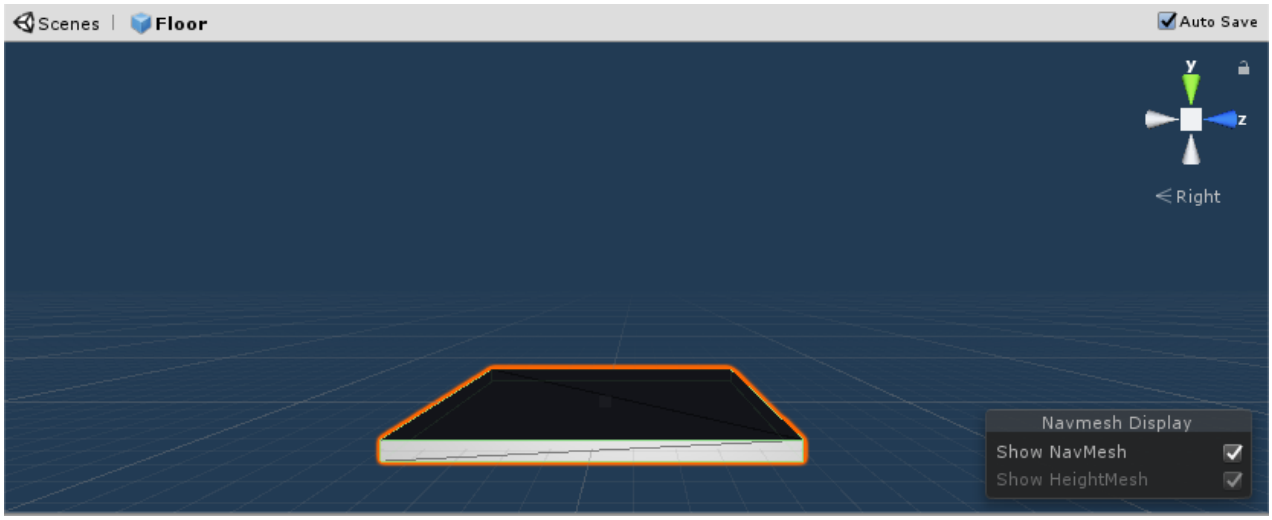


Рисунок 4.10 — Префаб підлоги

- Камер(рисунок 4.11)

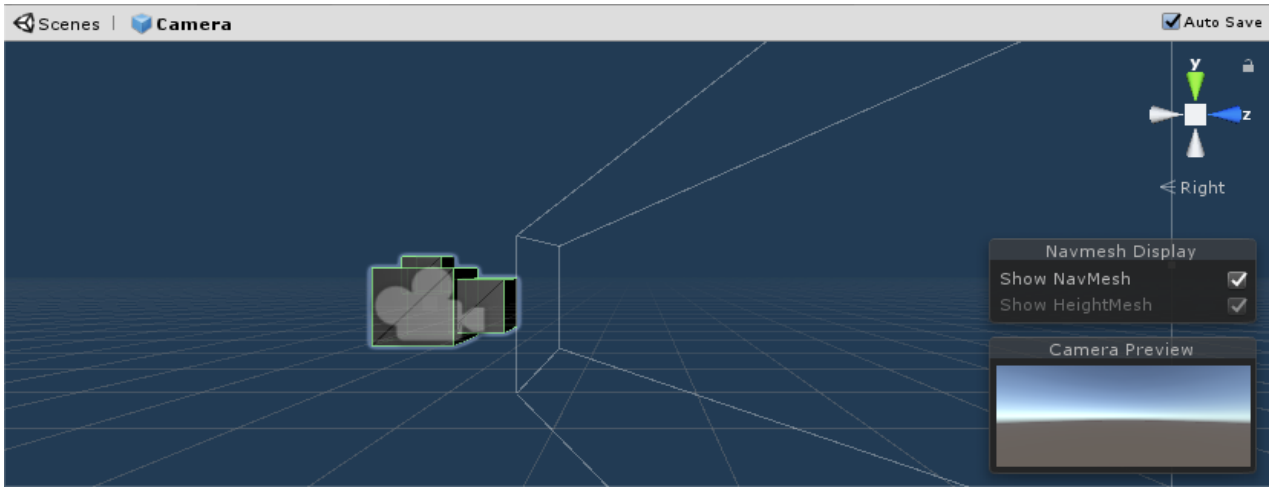


Рисунок 4.12 — Префаб камер

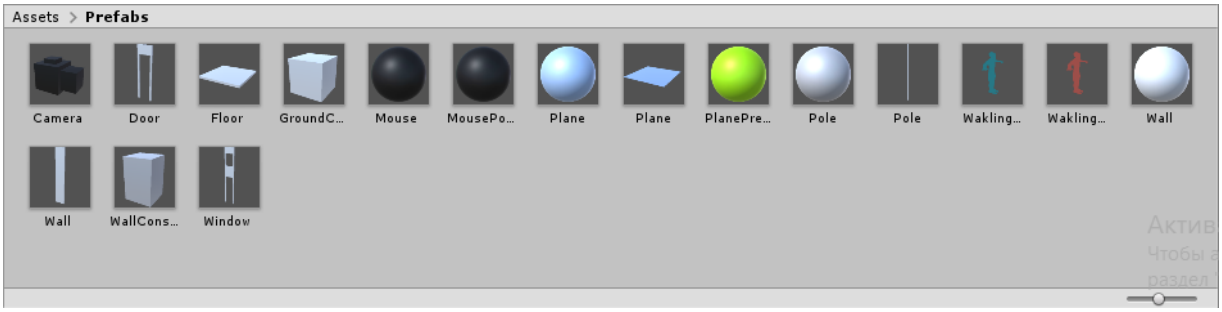


Рисунок 4.13 — Папка з створеними префабами та матеріалами

Наступний етап алгоритму полягає у зчитуванні вводу користувача, і, на основі цих даних, встановлення необхідного префабу з початкової точки вводу до точки кінцевого вводу. Було створено окремий клас MousePointer, який чекає на дію користувача і записує позицію курсора при натисканні на екран. Далі, ці данні передаються в клас конструктора, після чого будується відповідний префаб.

### 4.6 Розробка алгоритму завантаження 3D-моделі

Алгоритм завантаження 3D моделі, маючи лише шлях до нього, був реалізований за допомогою інструменту Unity AssetBundle.

Щоб створити AssetBundle, необхідно зсередини сценарію редактора викликати базовий метод BuildPipeline.BuildAssetBundles().

Далі алгоритм працює наступним чином: під час роботи програми, користувач вводить шлях до моделі необхідного приміщення. Ці данні одразу ж записуються до вбудованого файлу разом з деякими іншими параметрами. Таким чином, цей алгоритм завантажує файл, який пізніше динамічно завантажується під час виконання за допомогою базового методу AssetBundle.LoadAsset ().

### 4.7 Розробка алгоритму завантаження даних з інших програм

Реалізований алгоритм завантаження даних, експортованих з інших програм у форматі JSON, був створений за допомогою JSON серіалізації. Це такий метод класу

JsonUtility, який використовується для перетворення об'єктів Unity у формат Json і навпаки.

Для отримання необхідних даних з іншої програми, було застосовано десеріалізацію. Цей метод обробки інформації вимагає відповідний клас, який буде записувати дані локально. В цей же час, встановлюються ігрові об’єкти з інформацією, яка була збережена раніше. Таким чином, концепт цього алгоритму полягає у створенні нового об’єкту з такою ж самою позицією і поворотом, з якою вони були вказані у серіалізованому файлі.

### 4.8 Розробка алгоритму генерування VR-сцени

Автоматична генерація мешів для всіх елементів приміщення, на відміну від простого розташування заздалегідь створених моделей, забезпечує набагато більшу гнучкість додатку. Користувач меншим чином залежить від розробника та має майже повну свободу дій у програмі.

Було реалізовано авторський алгоритм, який генерує сцену з приміщенням, яке користувач побудував/завантажив самостійно. Система шукає необхідні ресурси Google VR SDK і вмикає режим демонстрації де це необхідно.

Використовуючи бібліотеку класів від Unity, було отримано доступ до всіх необхідних інструментів GVR, які відповідають за візуалізацію у VR. На основі цих інструментів було створено метод, який підтягує необхідні об’єкти до батьківського GVR об’єкту та візуалізовує їх в залежності від цілі спостереження.

### 4.9 Алгоритм NavMesh-запікання та запуску агентів

Перш ніж використовувати NavMesh в Unity, було відрегульовано кілька налаштувань. Було включено відображення вікон навігації, щоб мати доступ до всіх ресурсів NavMesh. Крім того, було створено префаб та анімацію руху агента(рисунок 4.14).

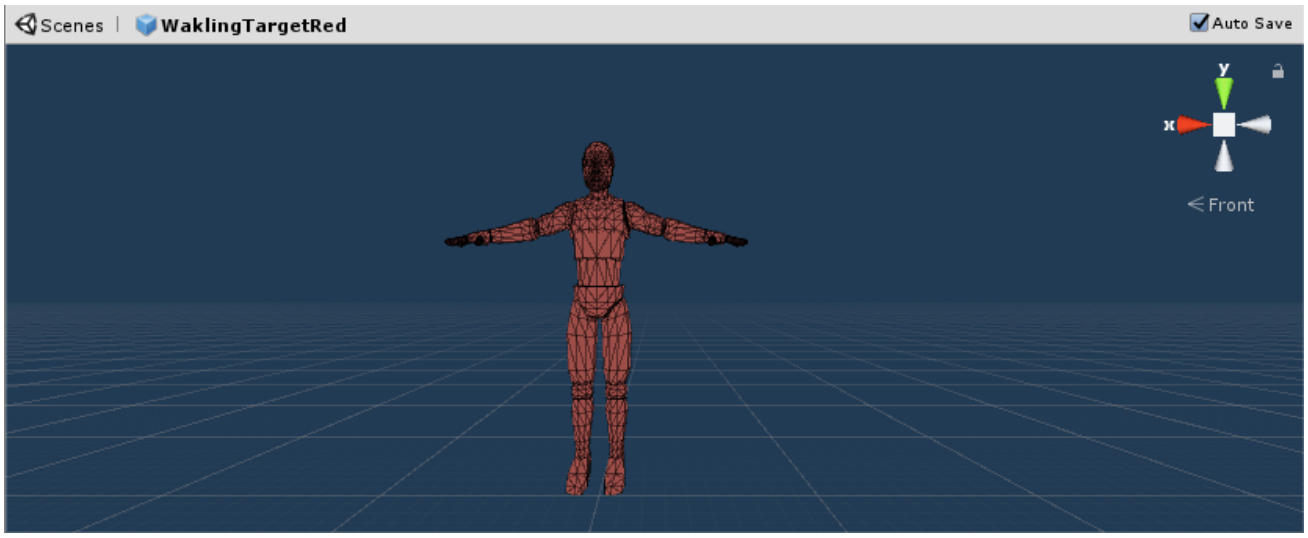


Рисунок 4.14 — Префаб агента

Далі було необхідно створити сценарій для дії NavMesh. У цьому сценарії статичні об'єкти мають діяти як перешкоди. Так як перешкоди мають бути створені або завантажені користувачем самостійно – було вирішено реалізувати розмежування об’єктів через компонент NavMesh Surface, який відповідає за запікання об’єктів у NavMesh. Цей компонент був налаштований та вкладений до всіх необхідних префабів та об’єктів(рисунок 4.15).

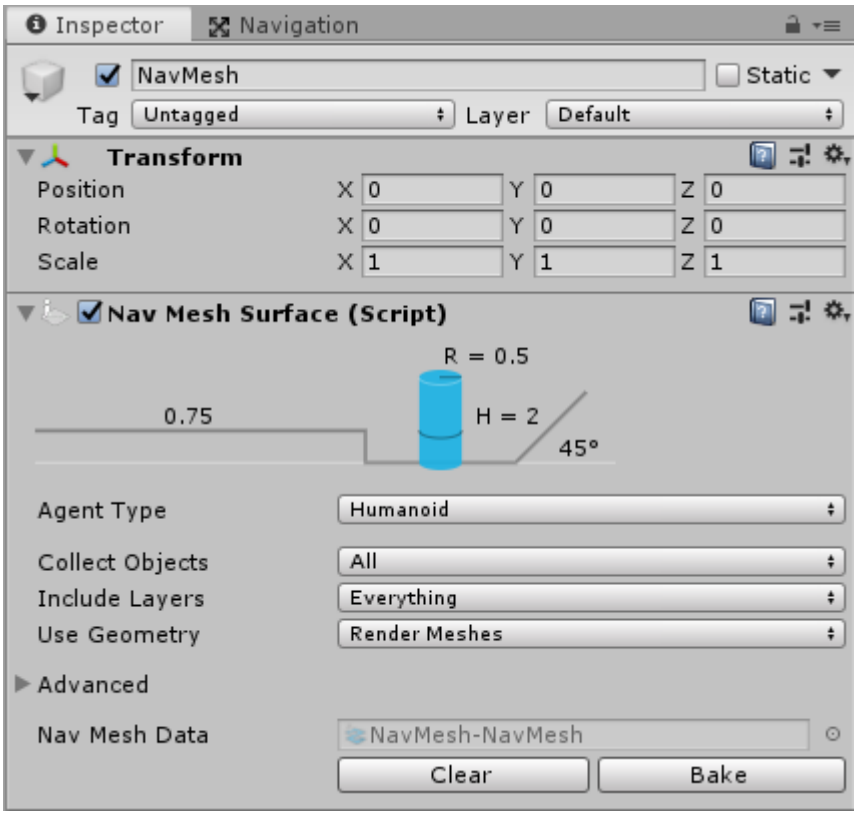


Рисунок 4.15 — Компонент NavMeshSurface

Після розмежування області NavMesh, перше що потрібно було зробити – налаштувати агентів, що будуть використовувати його. Характеристики, які було необхідно заповнити:

- Назва: ім'я агента.
- Радіус: радіус тіла агента.
- Висота: висота агента, це значення буде обмежувати максимальну висоту, де агент пройде, не застрягаючи.
- Висота кроку: значення, що вказує наскільки високою може бути місцевість, щоб агент міг піднятися вгору.
- Максимальний нахил: максимальний нахил, через який може пройти агент.

Останній етап алгоритму найважливіший – запікання. Після того, як користувач захоче згенерувати сцену, система має «запекти» NavMesh приміщення. Запропонований алгоритм  перебирає всі об’єкти з компонентом NavMeshSurface і запікає для агентів приміщення у межах реального часу. В цей час, агентам встановлюються точки-цілі, які вони мають досягнути, і маршрут з урахуванням специфіки приміщення.

**Висновок до розділу 4**

У результаті проектування та розробки системи автоматизованого генерування VR-сцен було створено додаток на платформі ігрового рушія. Запропонований програмний продукт здатний конструювати, завантажувати і генерувати VR-приміщення. Крім того, було створено алгоритм завантаження позиції та повороту об’єктів камери з JSON формату. Користувач може взаємодіяти з додатком через будь-яку платформу та може здійснювати контроль над додатком за допомогою інтерфейсу головного меню. Для демонстрації та інтеграції системи було обрано прикладну задачу виявлення сліпих зон у продуктовому магазині.

# 5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Запропонований програмний комплекс розроблений з використанням ігрового рушія Unity і тому може працювати на будь-яких платформах.

## 5.1 Інсталяція та системні вимоги

Враховуючи все вище сказане, досліджений програмний комплекс, розроблений на багатоплатформовому ігровому рушії здатен працювати в межах будь-якої операційної системи, окрім iOS, маючи при цьому мінімальні характеристики:

- Android – починаючи з версії 4.1
- Windows, Linux, та Mac – архітектурою x64 або x86.

## 5.2 Інструкція з використання програмного продукту

Першим етапом роботи користувача з клієнтським додатком є головне меню, де він має змогу налаштувати сцену-конструктор, завантажити або створити новий проект. На рисунку 5.1 представлене зображення головного меню.

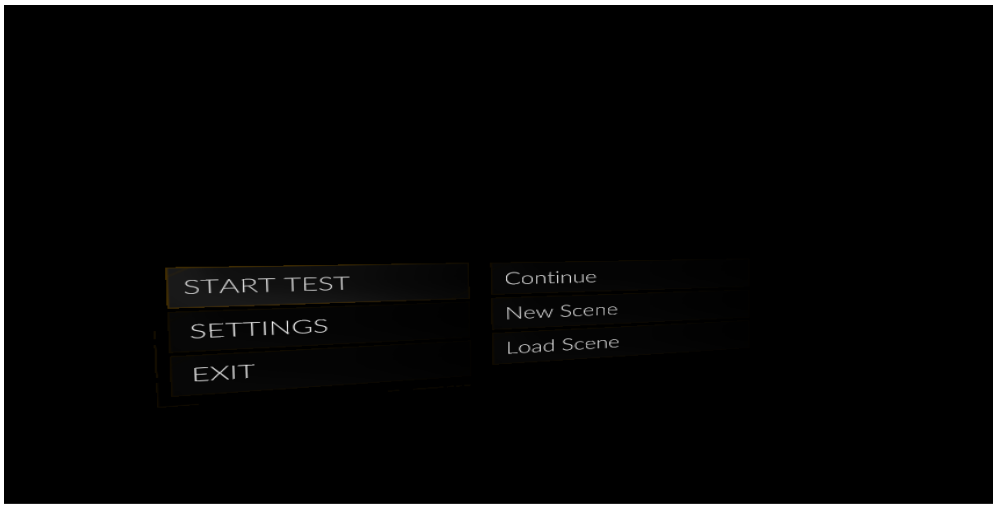


Рисунок 5.1 — Головне меню

На рисунку 5.2 представлене зображення налаштування з головного меню.


Рисунок 5.2 — Налаштування з головного меню

Після того, як користувач налаштував сцену-конструктор, прийняв рішення щодо завантаження свого проекту або створення нового, починається наступний етап – завантаження сцени-конструктора.

На рисунку 5.3 зображена головна сцена системи.


Рисунок 5.3 — Головна сцена системи

На головній сцені користувач має доступ до функціоналу конструювання або завантаження 3D-плану приміщення, встановлення камер, дослідження сцени та генерування VR-сцени.

Створюючи новий проект, користувач може будувати приміщення за допомогою запропонованих інструментів:

- Стін (Рисунок 5.4)

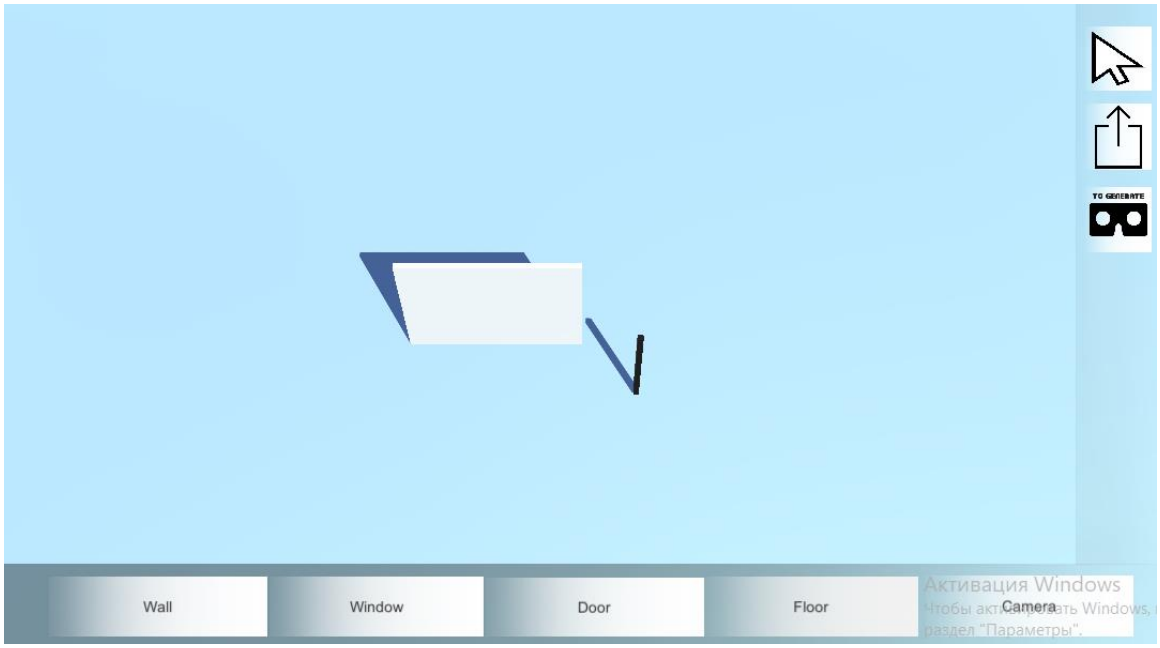


Рисунок 5.4 — Побудова стіни

- Вікон (Рисунок 5.5)

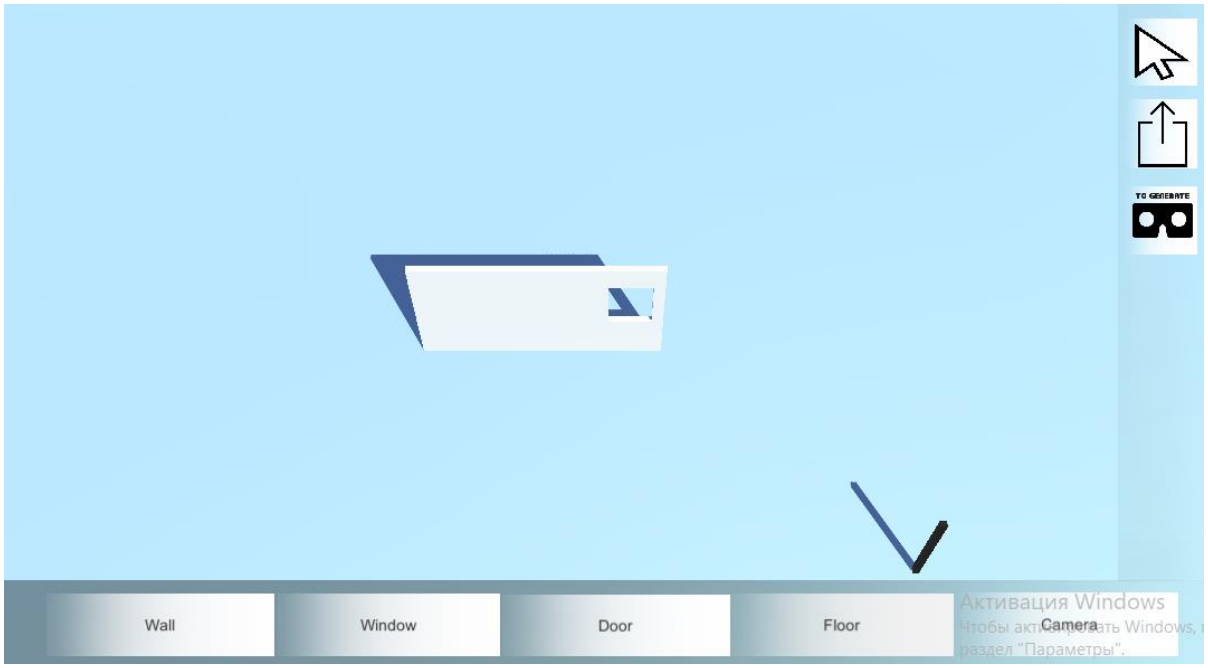


Рисунок 5.5 — Побудова вікон

- Дверей (Рисунок 5.6)

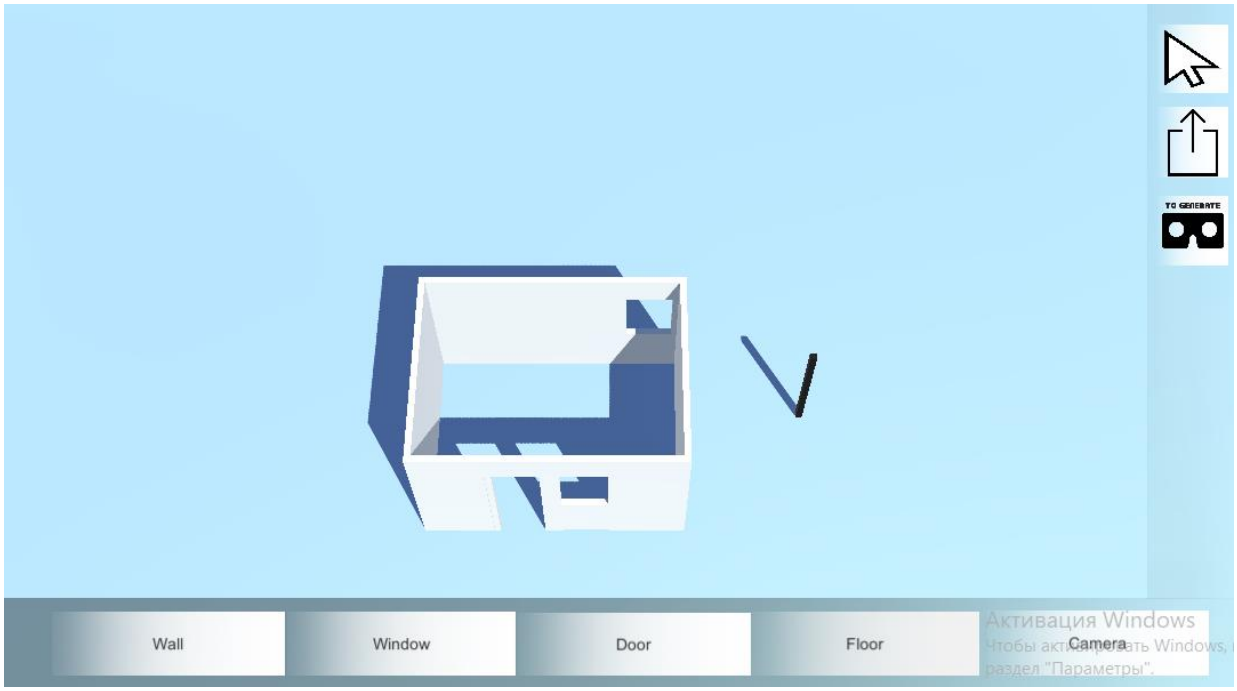


Рисунок 5.6 — Побудова дверей

- Підлоги (Рисунок 5.7)

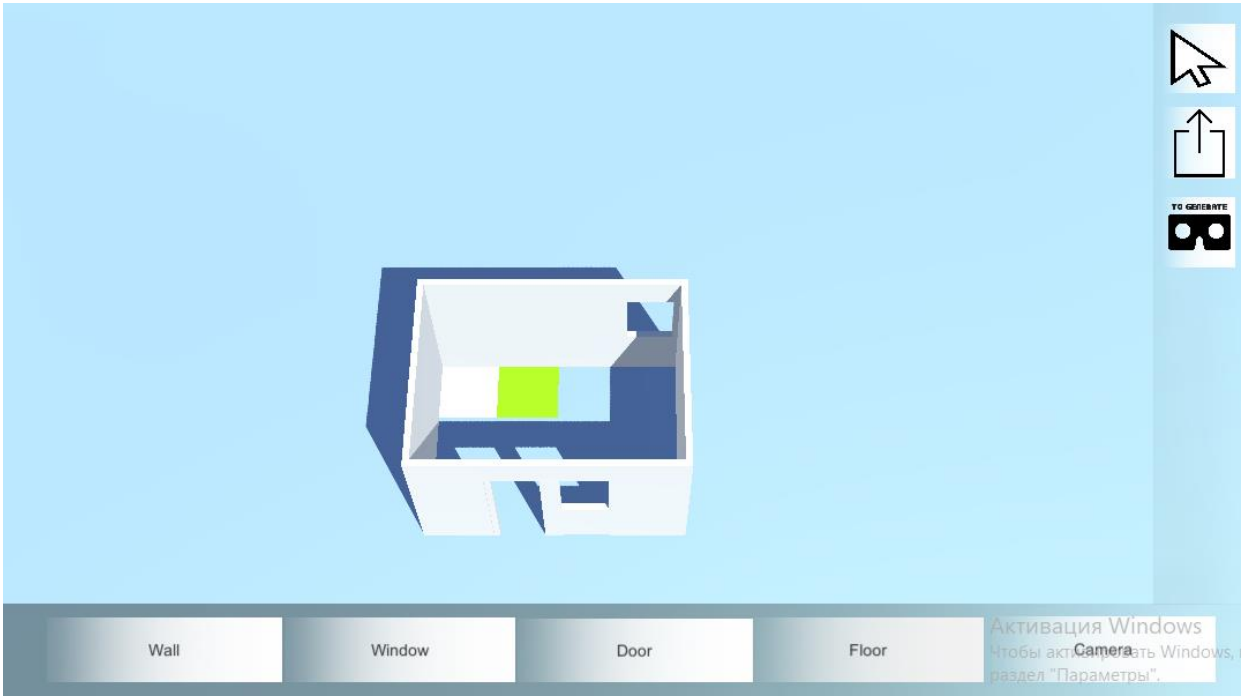


Рисунок 5.7 — Побудова підлоги

- Камери (Рисунок 5.8)

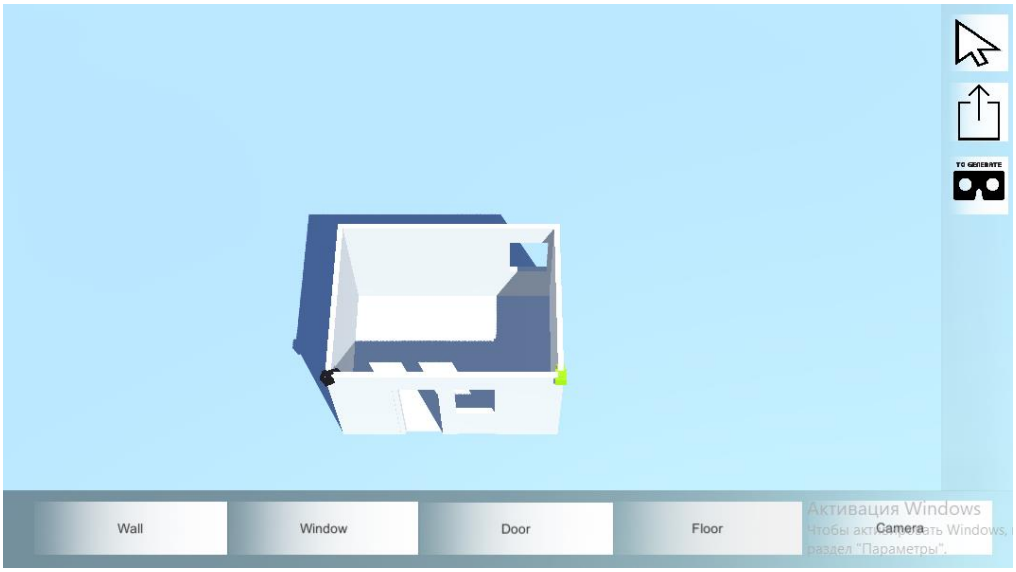


Рисунок 5.8 — Встановлення камер

У випадку, коли користувач завантажує власний проект, він виключає необхідність використання запропонованих інструментів. Крім того, завантажуючи проект, користувач може вказати шлях на JSON файл, який містить в собі інформацію про розміщення камер. Єдиною умовою є заповнення користувачем поля, де він має вказати шлях до розміщення файлу (Рисунок 5.9).

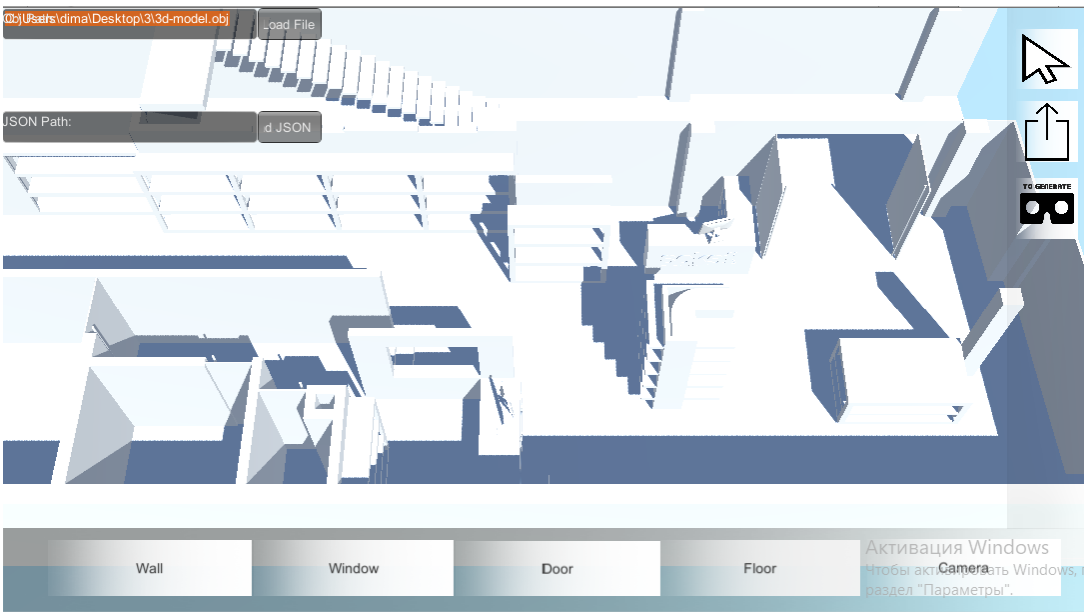


Рисунок 5.9 — Завантаження власного 3D-плану приміщення

Після завантаження власного 3D-плану, користувач має можливість встановити камери або завантажити їх з JSON файлу, задати точки для дослідження NavMesh-агентам та згенерувати VR-сцену, де можна перевірити приміщення на сліпі зони. Кожна встановлена камера, в свою чергу, буде візуалізовувати процес. (Рисунок 5.10).

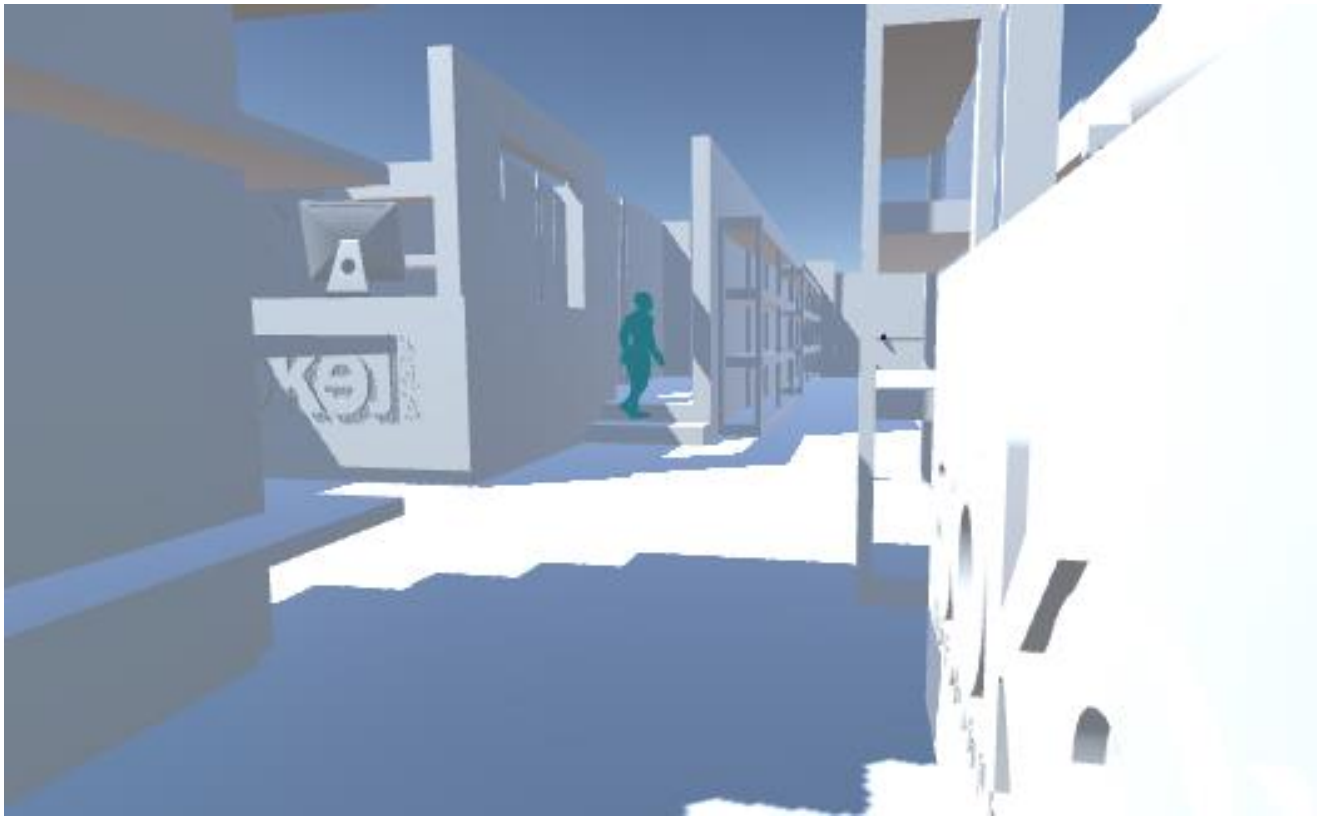

Рисунок 5.10 — Генерування та візуалізація VR-сцени

Користувачеві залишається лише спостерігати за процесом дослідження агентів. Після чого, він може зробити певні висновки щодо надійності даного приміщення.

## Висновок до розділу 5

В результаті було створено додаток, який демонструє інтеграцію запропонованої системи у прикладну сферу діяльності. Головними функціями даного програмного продукту є створення чи завантаження 3D плану приміщення,

завантаження заздалегідь підготовлених камер, генерація VR-сцени, візуалізація дослідження приміщення агентами.

# ВИСНОВКИ

Проаналізувавши аналоги даного програмного комплексу та дослідивши проблему автоматизації прикладних процесів, було виявлено, що більшість систем є складними для звичайного користувача та складаються з громіздкого функціоналу. Крім того, для використання побудованих моделей у прикладних задачах, необхідно застосовувати інструменти інших програмних комплексів. Тому було вирішено створити авторську систему автоматизованого генерування VR-сцен, яка б полегшила роботу з системою та дала змогу використовувати її на прикладних задачах.

В ході дослідження було поставлено наступне завдання: автоматизувати генерацію VR-сцени прикладного процесу(на прикладі продуктового магазину) для візуалізації та перевірки розставлених камер на сліпі зони. Дати змогу користувачеві конструювати або завантажувати модель приміщення, зчитувати данні з файлів, експортованих з інших програм.

Під час проектування програмного продукту був проведений аналіз методів та засобів розробки, обґрунтовано вибір платформи для розробки, відповідних інструментів реалізації та мови програмування.

За час виконання дипломної роботи, було розроблено запропонований програмний комплекс, що дало змогу покращити знання не тільки у використанні різноманітних технологій, але і у розробці алгоритмів на мові програмування C#. Варто зазначити, що в ході досліджень платформи Unity, було отримано досвід не лише у розробці звичайних додатків Unity, але і у використанні застосунків віртуальної реальності.

Під час перевірки тестових завдань на запропонованій системі, була підтверджена коректність отриманих результатів побудови, завантаження, генерації та візуалізації. Отже, система відповідає поставленим вимогам, поставлене завдання було виконано.

Розроблений програмний продукт є доступним для звичайного користувача адже майже всі процеси є автоматизованими, що дозволяє побудувати новий або завантажити власний 3D-план приміщення у VR-сцені.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Крейтон, P.X. Unity Game Development Essentials / P.X. Крейтон Packt Publishing, 2010, 83 с.

2. Павловская, Т.А. C#. Программирование на языке высокого уровня / Т.А. Павловская - Спб; ПИТЕР, 2009 432 с.

3. Шилдт, Г. C# Полное руководство / Г. Шилдт И.В. Берштейн Москва., Вильямс, 2011. - 1056 с.

4. Совертков, П. И. Занимательное компьютерное моделирование в элементарной математике. Учебное пособие / П.И. Совертков. - М.: Гелиос АРВ, 2004. - 384 с.

5. Скит, Джон C#. Программирование для профессионалов / Джон Скит. - М.: Вильямс, 2011. - 544 с.

6. Winn W.: Learning in Artificial Environments: Embodiment, Embeddedness and Dynamic Adaptation. Technology, Instruction, Cognition and Learning, 1(1), 87-114.-2003.

7. Linowes J.: Unity Virtual Reality Projects/ Linowes J.-2015.-286p.

8. Barrat, James. The latest invention of mankind / James Barrath. - M., 2015. – 299p.

9. Loscos, C., Widenfeld, H., Roussou, M. Meyer, A., Tecchia, F., Drettakis, G., Gallo, E.,Martinez, A. R., Tsingos, N., Chrysanthou, Y., Robert, L., Bergamasco, M., Dettori, A., & Soubra, S.: The CREATE Project: Mixed Reality for Design, Education, and Cultural Heritage with a Constructivist Approach, ISMAR 03, The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, The National Center of Sciences, Tokyo, Japan, Oct. 7 - Oct. 10 (2003).

10. Платов, В. Я. Деловые игры. Разработка, организация, проведение. Учебник / В.Я. Платов. - М.: Профиздат, 2010. - 192 с.

11. Любанова, Т.П. Бизнес-план: опыт, проблемы. Содержание бизнес-плана, пример разработки / Т.П. Любанова, Л.В. Мясоедова, Т.А. Грамотенко, и др.. - М.: Приор, 2012. - 204 с.

12. Хорхе, Паласиос Unity 5.x. Программирование искусственного интеллекта в играх. Руководство / Паласиос Хорхе. - М.: ДМК Пресс, 2017. - 427 с.

13. Алгазинов, Э. К. Анализ и компьютерное моделирование информационных процессов и систем / Э.К. Алгазинов, А.А. Сирота. - М.: Диалог-Мифи, 2009. - 416 с.

14. Kim, J. Modeling and Optimization of a Tree Based on Virtual Reality for Immersive Virtual Landscape Generation. Symmetry 2016, 8, 93.

15. Slater, M.; Usoh, M. Simulating peripheral vision in immersive virtual environments. Comput. Graph. 1993, 17, 643–653.

16. Jeong, K.; Lee, J.; Kim, J. A Study on New Virtual Reality System in Maze Terrain. Int. J. Hum. Comput. Interact. 2018, 34, 129–145.

17. Goldstone W. Unity Game Development Essentials. / W Goldstone. Birmingham: Packt Publishing Ltd., — 2009. — 316 c.

# ДОДАТОК А

Система автоматизованого генерування VR-сцен

## Специфікація

Аркушів 2

2020

| Позначення | Найменування | Примітки |
|---|---|---|
| Документація | | |
| УКР.НТУУ“КПІ”.ТВ6150_20Б 81-1 | Записка | Пояснювальна записка |
| Компоненти | | |
| УКР.НТУУ«КПІ».ТВ6150_20Б 12-1 | Текст програмного модулю | Частина додатку, що відповідає за пошук позиції вводу користувача |
| УКР.НТУУ«КПІ».ТВ6150_20Б 12-2 | Текст програмного модулю | Частина програмного коду створення сегменту стіни |
| УКР.НТУУ«КПІ».ТВ6150_20Б 12-3 | Текст програмного модулю | Частина додатку, що відповідає за читання та завантаження моделі |
| УКР.НТУУ«КПІ».ТВ6150_20Б 12-4 | Текст програмного модулю | Частина додатку, що відповідає за читання та завантаження json файлу |
| УКР.НТУУ«КПІ».ТВ6150_20Б 12-5 | Текст програмного модулю | Встановлення агента-досліджувача шляху |
| УКР.НТУУ«КПІ».ТВ6150_20Б 13-1 | Опис програмного модулю | Опис програмного модулю |

# ДОДАТОК Б

Система автоматизованого генерування VR-сцен

## Текст програмного модулю

Аркушів 6

2020

```
//MOUSE POSITION
void Update()
    {
        mousePointer.transform.position=snapPosition(getWorldPoint());
        anotherMousePointer.transform.position=snapPosition(getWorldPoint());
        cameraPointer.transform.position=snapPosition(getWorldPoint());
        handPointer.transform.position=snapPosition(getWorldPoint());
    }

    public Vector3 getWorldPoint()
    {
        Camera cam = GetComponent<Camera>();
        Ray ray = cam.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if(Physics.Raycast(ray, out hit))
        {
            if(floor==true)
                return hit.point*2;
            else if(camera==true)
                return hit.point*3;
            else return hit.point;
        }
        return Vector3.zero;
    }
//CREATE WALL
void createWallSegment(Vector3 current)
    {
        GameObject newPole = Instantiate(polePrefab,current,Quaternion.identity);
```

```
        Vector3 middle =
Vector3.Lerp(newPole.transform.position,lastPole.transform.position,0.5f);
        GameObject newWall = Instantiate(wallPrefab,middle,Quaternion.identity);
        newWall.transform.LookAt(lastPole.transform);
        Global.GlobalAddToArrayList(newPole);
        Global.GlobalAddToArrayList(newWall);
        DontDestroyOnLoad(newPole);
        DontDestroyOnLoad(newWall);
        lastPole=newPole;
    }
private void OnGUI() {
        objPath = GUI.TextField(new Rect(0, 0, 256, 32), objPath);
        GUI.Label(new Rect(0, 0, 256, 32), "Obj Path:");

        jsonPath = GUI.TextField(new Rect(0, 104, 256, 32), jsonPath);
        GUI.Label(new Rect(0, 104, 256, 32), "JSON Path:");
//LOAD FILE
        if(GUI.Button(new Rect(256, 0, 64, 32), "Load File"))
        {
            //file path
            if (!File.Exists(objPath))
            {
                error = "File doesn't exist.";
            }else{
                if(loadedObject != null)
                    Destroy(loadedObject);
                loadedObject = new OBJLoader().Load(objPath);
                Vector3 scale = new Vector3(0.025f,0.025f,0.025f);
                loadedObject.transform.localScale = scale;
```

```
            loadedObject.AddComponent<NavMeshSurface>();
            loadedObject.name="Empty(OBJ)";
            Global.GlobalAddToArrayList(loadedObject);
            DontDestroyOnLoad(loadedObject);
            error = string.Empty;
        }
    }
//LOAD JSON
    if(GUI.Button(new Rect(256, 104, 64, 32), "Load JSON File"))
    {
        if (!File.Exists(jsonPath))
        {
            error = "File doesn't exist.";
        }
        else
        {
            JSONstring = File.ReadAllText (jsonPath);
            CameraDataListObject cameras = new CameraDataListObject();
            cameras = JsonUtility.FromJson<CameraDataListObject>(JSONstring);
            foreach (CameraData camera in cameras.cameraList)
            {
                xp=camera.transformX;
                yp=camera.transformY;
                zp=camera.transformZ;
                xr=camera.rotationX;
                yr=camera.rotationY;
                zr=camera.rotationZ;
                Vector3 position = new Vector3(xp,yp,zp);
```

```
                Vector3 rotationVector = new Vector3(xr,yr,zr);
                Quaternion rotation = Quaternion.Euler(rotationVector);
                loadedCamera = Instantiate(cameraPrefab,position,rotation);
                Debug.Log(camera.transformX);
                Camera cameraInChildren =
loadedCamera.GetComponentInChildren<Camera>();
                cameraInChildren.enabled = false;
                Global.GlobalAddToArrayList(loadedCamera);
                DontDestroyOnLoad(loadedCamera);
                error = string.Empty;
            }
        }
    }
//TargetManager
void Start()
   {
      foreach (GameObject item in allObjects)
      {
         if(i<walkingTargetWayPoints.Length)
         {
            if(item.name=="Floor(Clone)(Clone)")
            {
               walkingTargetWayPoints[i]=item.GetComponent<Transform>();
               walkingTargetResp = item.GetComponent<Transform>();
               i++;
            }
         }
      }
```

```
        GameObject walkingTarget =
Instantiate(walkingTargetPrefab,walkingTargetResp.position,walkingTargetResp.rotation)
;
        walkingTarget.GetComponent<WalkingTarget>().Init(walkingTargetWayPoints);
        GameObject walkingTargetRed =
Instantiate(walkingPrefab,walkingTargetResp.position,walkingTargetResp.rotation);

walkingTargetRed.GetComponent<WalkingTargetRed>().Init(walkingTargetWayPoints);
    }
```

# ДОДАТОК В

Система автоматизованого генерування VR-сцен

## Опис програмного модулю

Аркушів 5

2020

# АНОТАЦІЯ

Метою дослідження є аналіз існуючих програмних застосунків в області автоматизованого генерування сцен, розробці засобів конструювання або завантаження 3D моделей приміщень та демонстрації дослідження цих моделей у VR-сцені штучним інтелектом.

# ЗМІСТ

# 1. ЗАГАЛЬНІ ВІДОМОСТІ

Вищеописаний програмний модуль створений у MVS Code 2019 як C# скрипти, прикріплені до ігрових об’єктів, створених у Unity Editor. Призначений для отримання даних про ввід користувача, а саме: вибір побудови елементу приміщення, вибір функціоналу сцени, шлях до моделі та json файлу. У модулі використана бібліотека для роботи з редактором – UnityEngine та бібліотека для роботи з штучним інтелектом та NavMesh компонентами – UnityEngine.AI.

## 1.1 Опис логічної структури

Даний програмний модуль описує концепт конструювання та завантаження моделей приміщення.

Початковим варіантом створення моделі у додатку – конструювання префабами. Програма зчитує ввід користувача, передає його підмодулю конструювання і встановлює префаб елементу приміщення з визначеними початковими та кінцевими точками.

Альтернативним варіантом отримання моделей у додатку – завантаження моделі з файлового потоку. Після отримання шляху до файлу .obj чи .json, контроллер визначає тип файлу та оброблює відповідно ці дані. Після обробки даних, створюються ігрові об’єкти.

Після чого користувач має змогу згенерувати VR сцену з відповідною моделлю, запустити агентів і почати дослідження цього приміщення.

## 1.2 Вхідні та вихидні дані

Вхідними даними до програмного модулю є ввід, присланий з клієнтського інтерфейсу. Вихідними даними є префаби, ігрові об’єкти та сцени.

## 1.3 Використані технічні засоби

При роботі програмного модулю використовувався комп’ютер на операцій системі Windows 10 з процесором Intel Core i7 та 16Гб оперативної пам’яті. Розроблена програма не залежить від платформи завдяки реалізації на Unity. Тому, вона може бути використана на будь-якій платформі.